

The software architecture

based on [Domain Driven Design] principles and patterns

Contents

1	Quick Start	9
1.1	With single-layer solution.....	9
1.2	With layered architecture.....	27
2	Getting Started	45
2.1	Web Application - Layered Architecture.....	45
2.1.1	1: Setup Your Development Environment	45
2.1.2	2: Creating a New Solution	46
2.1.3	3: Running the Solution	48
2.2	Web Application - Single-Layered Architecture.....	53
2.2.1	1: Setup Your Development Environment	54
2.2.2	2: Creating a New Solution	55
2.2.3	3: Running the Solution	56
2.3	Console Application.....	58
2.4	WPF Application	59
2.5	MAUI.....	59
2.6	Empty Web Project.....	60
3	Tutorials	63
3.1	Web Application Development.....	63
3.1.1	1: Creating the Server Side	63
3.1.2	2: The Book List Page	73
3.1.3	3: Creating, Updating and Deleting Books.....	88
3.1.4	4: Integration Tests	118
3.1.5	5: Authorization	123
3.1.6	6: Authors: Domain layer	135
3.1.7	7: Authors: Database Integration	141
3.1.8	8: Authors: Application Layer	146
3.1.9	9: Authors: User Interface	157
3.1.10	10: Book to Author Relation	181
3.2	Community Articles	205
3.3	Migrating from the ASP.NET Boilerplate.....	205
4	CLI	224
4.1	Examples for the new command	238

5	Startup Templates.....	244
5.1	Overall.....	244
5.2	Application.....	245
5.3	Application (Single Layer)	256
5.4	Module	259
5.5	Console	265
5.6	WPF.....	266
6	Fundamentals	267
6.1	Application Startup.....	267
6.2	Authorization.....	276
6.3	Caching	287
6.3.1	Entity Cache.....	294
6.3.2	Redis Cache.....	297
6.4	Configuration.....	298
6.5	Connection Strings.....	298
6.6	Dependency Injection.....	302
6.6.1	AutoFac Integration	313
6.7	Exception Handling	316
6.8	Localization	323
6.9	Logging.....	328
6.10	Object Extensions	329
6.11	Options	337
6.12	Settings	340
6.13	Validation.....	345
6.13.1	FluentValidation Integration.....	349
7	Infrastructure.....	351
7.1	Audit Logging.....	351
7.2	Background Jobs.....	360
7.2.1	Hangfire Integration	366
7.2.2	RabbitMQ Integration.....	370
7.2.3	Quartz Integration	373
7.3	Background Workers	377
7.3.1	Quartz Integration	380

7.3.2	Hangfire Integration	384
7.4	BLOB Storing	387
7.4.1	BLOB Storing System.....	387
7.4.2	Storage Providers.....	395
7.5	Cancellation Token Provider	411
7.6	CSRF/XSRF & Anti Forgery	413
7.7	Concurrency Check	417
7.8	Current User	420
7.9	Data Filtering	424
7.10	Data Seeding.....	430
7.11	Distributed Locking.....	435
7.12	Email Sending	435
7.12.1	Email Sending System.....	435
7.12.2	MailKit Integration.....	441
7.13	Event Bus	442
7.13.1	Overall.....	442
7.13.2	Local Event Bus	442
7.13.3	Distributed Event Bus	448
7.14	Features	476
7.15	Global Features.....	486
7.16	GUID Generation	489
7.17	Image Manipulation	492
7.18	JSON.....	500
7.19	Object to Object Mapping	502
7.20	Simple State Checker	509
7.21	SMS Sending	512
7.22	String Encryption	514
7.23	Text Templating.....	517
7.23.1	Razor Integration	518
7.23.2	Scriban Integration	529
7.24	Timing	540
7.25	Virtual File System	543
8	Architecture	547

8.1	Modularity	547
8.1.1	Basics	547
8.1.2	Plug-In Modules.....	552
8.1.3	Best Practices.....	558
8.1.4	Customizing/Extending Modules.....	579
8.2	Domain Driven Design	606
8.2.1	Overall.....	606
8.2.2	Domain Layer.....	607
8.2.3	Application Layer	638
8.2.4	E-Book: Implementing DDD	665
8.3	Multi Tenancy	665
8.4	Microservices.....	674
9	API.....	675
9.1	ABP Endpoints	675
9.1.1	Application Configuration.....	675
9.1.2	Application Localization	676
9.2	API Versioning.....	677
9.3	Auto API Controllers	684
9.4	Dynamic C# API Clients.....	689
9.5	Integration Services	694
9.6	Static C# API Clients	697
9.7	Swagger Integration.....	703
10	User Interface	707
10.1	MVC / Razor Pages.....	707
10.1.1	Overall.....	707
10.1.2	Navigation / Menus	711
10.1.3	Forms & Validation	718
10.1.4	Modals.....	723
10.1.5	Data Tables	734
10.1.6	Auto-Complete Select.....	741
10.1.7	Page Alerts.....	743
10.1.8	Dynamic JavaScript API Client Proxies	745
10.1.9	Static JavaScript API Client Proxies	747

10.1.10	Client Side Package Management	751
10.1.11	Bundling & Minification.....	754
10.1.12	Tag Helpers	764
10.1.13	Widgets.....	781
10.1.14	Toolbars	792
10.1.15	Page Header.....	794
10.1.16	Branding.....	796
10.1.17	Layout Hooks	797
10.1.18	Testing.....	800
10.1.19	Theming.....	804
10.1.20	JavaScript API.....	824
10.1.21	Customize/Extend the UI.....	846
10.1.22	Security	867
10.2	Blazor	870
10.2.1	Overall.....	870
10.2.2	Navigation / Menu.....	874
10.2.3	Localization	880
10.2.4	Theming.....	882
10.2.5	Security	910
10.2.6	Services.....	912
10.2.7	Other Components	921
10.2.8	Settings	922
10.2.9	Error Handling.....	923
10.2.10	Customization / Overriding Components	926
10.2.11	Global Scripts & Styles	928
10.2.12	Global Features.....	930
10.2.13	Routing.....	931
10.2.14	PWA Configuration	932
10.2.15	Layout Hooks	938
10.3	Angular	941
10.3.1	Quick Start	941
10.3.2	Development	948
10.3.3	Core Functionality.....	974

10.3.4	Utilities.....	1011
10.3.5	Customization	1065
10.3.6	Components	1139
10.4	React Native.....	1153
10.4.1	Getting Started	1153
10.5	Common	1157
11	Data Access.....	1189
11.1	Overall.....	1189
11.2	Entity Framework Core	1190
11.2.1	Database Migrations.....	1211
11.2.2	Switch DBMS.....	1225
11.3	MongoDB.....	1237
11.4	Dapper	1247
12	Real Time	1250
12.1	SignalR Integration.....	1250
13	Dapr Integration	1255
14	Testing.....	1268
15	Deployment	1285
15.1	Configuring OpenIddict	1286
15.2	Configuring for Production	1288
15.3	Deploying to a Clustered Environment.....	1291
15.4	Distributed / Microservice Solutions	1296
15.5	Optimizing for Production	1298
16	Application Modules.....	1299
16.1	Overall.....	1299
16.2	Account.....	1300
16.3	Audit Logging.....	1302
16.4	Background Jobs.....	1304
16.5	CMS Kit	1305
16.6	Docs	1309
16.7	Feature Management	1324
16.8	Identity	1327
16.9	IdentityServer	1335

16.9.1	IdentityServer Migration Guide	1339
16.10	OpenIddict	1344
16.10.1	OpenIddict Migration Guide.....	1347
16.11	Permission Management.....	1353
16.12	Setting Management	1355
16.13	Tenant Management	1362
16.14	Virtual File Explorer	1365
16.15	Common	1367
16.15.1	Database Tables	1367
17	Samples	1382
17.1	All Samples	1382
17.2	eShopOnAbp.....	1384
17.3	EventHub	1384
17.4	Microservice Demo (legacy)	1385
18	Books	1419
18.1	Mastering ABP Framework.....	1419
18.2	Implementing Domain Driven Design.....	1419
19	Release Information	1419
19.1	Upgrading	1419
19.2	Official Packages	1421
19.3	Preview Releases	1421
19.4	Nightly Builds.....	1423
19.5	Road Map	1423
19.6	Migration Guides	1425
20	API Documentation.....	1425
21	Contribution Guide.....	1425

1 Quick Start

Quick Start: Overall

Welcome to the ABP Framework. This is a single-part, quick-start tutorial to build a simple application. Start with this tutorial if you want to quickly understand how ABP Framework works.

Select the Solution Architecture

This tutorial has multiple versions. Please select the one best fits for you:

- * **[Single-Layer Solution](Single-Layer/Index.md)**: Creates a single-project solution. Recommended for building an application with a **simpler and easy to understand** architecture.
- * **[Layered Solution Architecture](Index.md)**: A fully layered (multiple projects) solution based on [Domain Driven Design](../../Domain-Driven-Design.md) practices. Recommended for long-term projects that need a **maintainable and extensible** codebase.

See Also

- * Check the [Web Application Development Tutorial](../Part-1.md) to see a real-life web application development in a layered architecture.

1.1 With single-layer solution

Quick Start

```
```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}
...```

```

This is a single-part quick-start tutorial to build a simple todo application with the ABP Framework. Here's a screenshot from the final application:

![todo-list](../todo-list.png)

You can find the source code of the completed application [here](<https://github.com/abpframework/abp-samples/tree/master/TodoApp-SingleLayer>).

```
{{if UI=="Blazor"}}
We are currently preparing a video tutorial for Blazor UI. You can watch other tutorials for the three UI types from
[here](https://www.youtube.com/playlist?list=PLsNclT2aHJcPqZxk7D4tU8LtTeCFcN_ci).
{{else}}
This documentation has a video tutorial on **YouTube**!! You can watch it here:
{{end}}```

```

```

{{if UI=="MVC" && DB == "EF"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/Z6jZSPB19iw" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

{{else if UI=="BlazorServer" && DB=="EF"}}

<iframe width="560" height="315" src="https://www.youtube.com/embed/-ynMYXBIg4Q" title="YouTube video player" frameborder="0"
allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope;
picture-in-picture; web-share" allowfullscreen></iframe>

{{else if UI=="NG" && DB=="EF"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/Pz4YWsU7CUs" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

{{else if UI=="MVC" && DB=="Mongo"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/i9oDVL1J7Dk" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

{{else if UI=="BlazorServer" && DB=="Mongo"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/z7YGDjcsTTs" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

{{else if UI=="NG" && DB=="Mongo"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/LdKLIHi9S8I" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>

{{end}}

Pre-Requirements

* An IDE (e.g. [Visual Studio](https://visualstudio.microsoft.com/vs/)) that supports [.NET 7.0+](https://dotnet.microsoft.com/download/dotnet) development.
* [Node v16.x](https://nodejs.org/)

{{if DB=="Mongo"}}

* [MongoDB Server 4.0+](https://docs.mongodb.com/manual/administration/install-community/)

{{end}}

```

## ## Creating a New Solution

In this tutorial, we will use the [ABP CLI](../../../../CLI.md) to create the sample application with the ABP Framework. You can run the following command in a command-line terminal to install the \*\*ABP CLI\*\*, if you haven't installed it yet:

```
```bash
dotnet tool install -g Volo.Abp.Cli
````
```

Then create an empty folder, open a command-line terminal and execute the following command in the terminal:

```
```bash
abp new TodoApp -t app-nolayers{{if UI=="BlazorServer"}} -u blazor-
server{{else if UI=="Blazor"}} -u blazor{{else if UI=="NG"}} -u
angular{{end}}{{if DB=="Mongo"}} -d mongodb{{end}}
````
```

```
 {{if UI=="NG"}}
```

This will create a new solution, named *\*TodoApp\**, with `'angular'` and `'aspnet-core'` folders. Once the solution is ready, open the solution (in the `'aspnet-core'` folder) with your favorite IDE.

```
 {{else if UI=="Blazor"}}
```

This will create a new solution with three projects:

- \* A `'blazor'` application that contains the Blazor code, the client-side.
- \* A `'host'` application, hosts and serves the `'blazor'` application.
- \* A `'contracts'` project, shared library between these two projects.

Once the solution is ready, open it in your favorite IDE.

```
 {{else}}
```

This will create a new solution with a single project, named *\*TodoApp\**. Once the solution is ready, open it in your favorite IDE.

```
 {{end}}
```

## ### Create the Database

You can run the following command in the  `{{if UI=="Blazor"}}` directory of your `'TodoApp.Host'` project  `{{else}}root` directory of your project (in the same folder of the `'.csproj'` file) `{{end}}` to create the database and seed the initial data:

```
```bash
dotnet run --migrate-database
````
```

This command will create the database and seed the initial data for you. Then you can run the application.

```
Run the Application
```

```
{{if UI=="MVC" || UI=="BlazorServer"}}
```

It is good to run the application before starting the development. Running the application is pretty straight-forward, you can run the application with any IDE that supports .NET or by running the `'dotnet run'` CLI command in the directory of your project:

```
{{else if UI=="Blazor"}}
```

It is good to run the application before starting the development. Running the application is pretty straight-forward, you just need to run the `'TodoApp.Host'` application with any IDE that supports .NET or by running the `'dotnet run'` CLI command in the directory of your project.

> \*\*Note:\*\* The `'host'` application hosts and serves the `'blazor'` application. Therefore, you should run the `'host'` application only.

After the application runs, open the application in your default browser:

```
{{else if UI=="NG"}}
```

It is good to run the application before starting the development. The solution has two main applications:

- \* `'TodoApp'` (in the .NET solution) hosts the server-side HTTP API, so the Angular application can consume it. (server-side application)
- \* `'angular'` folder contains the Angular application. (client-side application)

Firstly, run the `'TodoApp'` project in your favorite IDE (or run the `'dotnet run'` CLI command on your project directory) to see the server-side HTTP API on [Swagger UI](<https://swagger.io/tools/swagger-ui/>):

```
![todo-swagger-ui-initial](./todo-single-layer-ui-initial.png)
```

You can explore and test your HTTP API with this UI. If it works, then we can run the Angular client application.

You can run the application using the following (or `'yarn start'`) command:

```
```bash
npm start
```
```

This command takes time, but eventually runs and opens the application in your default browser:

```
{{end}}
```

```
![todo-ui-initial](../todo-ui-initial.png)
```

You can click on the `*Login*` button and use `'admin'` as the username and `'1q2w3E*'` as the password to login to the application.

All right. We can start coding!

```
Defining the Entity
```

This application will have a single [entity](../../../../Entities.md) and we can start by creating it. So, create a new `TodoItem` class under the `Entities` folder of {{if UI=="Blazor"}}`the `TodoApp.Host` project{{else}}`the project{{end}}:

```
```csharp
using Volo.Abp.Domain.Entities;

namespace TodoApp{{if UI=="Blazor"}}.{{end}}Entities;

public class TodoItem : BasicAggregateRoot<Guid>
{
    public string Text { get; set; }
}
````
```

`BasicAggregateRoot` is the simplest base class to create root entities, and `Guid` is the primary key (`Id`) of the entity here.

```
Database Integration
```

```
{{if DB=="EF"}}
```

Next step is to setup the [Entity Framework Core](../../../../Entity-Framework-Core.md) configuration.

```
Mapping Configuration
```

Open the `TodoAppDbContext` class (in the `Data` folder) and add a new `DbSet` property to this class:

```
```csharp
public DbSet<TodoItem> TodoItems { get; set; }
````
```

Then navigate to the `OnModelCreating` method in the same class and add the following mapping code for the `TodoItem` entity:

```
```csharp
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    /* Include modules to your migration db context */

    builder.ConfigurePermissionManagement();
    ...

    /* Configure your own tables/entities inside here */
    builder.Entity<TodoItem>(b =>
    {
        b.ToTable("TodoItems");
    });
}
````
```

We've mapped the `TodoItem` entity to the `TodoItems` table in the database. The next step is to create a migration and apply the changes to the database.

### ### Code First Migrations

The startup solution is configured to use Entity Framework Core [Code First Migrations](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations>). Since we've changed the database mapping configuration, we should create a new migration and apply changes to the database.

Open a command-line terminal in the {{if UI=="Blazor"}} directory of your `TodoApp.Host` project {{else}}root directory of your project (in the same folder of the `.csproj` file){{end}} and type the following command:

```
```bash
dotnet ef migrations add Added_TodoItem
````
```

This will add a new migration class to the project. You should see the new migration in the `Migrations` folder:

![todo-efcore-migration](todo-efcore-migration-single-layer.png)

Then, you can apply changes to the database using the following command, in the same command-line terminal:

```
```bash
dotnet ef database update
````
```

{{else if DB=="Mongo"}}

The next step is to setup the [MongoDB](../../../../../MongoDB.md) configuration. Open the `TodoAppDbContext` class (under the \*\*Data\*\* folder) in your project and make the following changes:

1. Add a new property to the class:

```
```csharp
public IMongoCollection<TodoItem> TodoItems => Collection<TodoItem>();
````
```

2. Add the following code inside the `CreateModel` method:

```
```csharp
modelBuilder.Entity<TodoItem>(b =>
{
    b.CollectionName = "TodoItems";
});
````
```

{{end}}

After the database integrations, now we can start to create application service methods and implement our use-cases.

## ## Creating the Application Service

An [application service](../../Application-Services.md) is used to perform the use cases of the application. We need to perform the following use cases in this application:

- \* Get the list of the todo items
- \* Create a new todo item
- \* Delete an existing todo item

Before starting to implement these use cases, first we need to create a DTO class that will be used in the application service.

### ### Creating the Data Transfer Object (DTO)

[Application services](../../Application-Services.md) typically get and return DTOs ([Data Transfer Objects](../../Data-Transfer-Objects.md)) instead of entities. So, create a new `TodoItemDto` class under the `Services/Dtos` folder{{if UI=="Blazor"}} of your `TodoApp.Contracts` project{{end}}:

```
```csharp
namespace TodoApp.Services.Dtos;

public class TodoItemDto
{
    public Guid Id { get; set; }
    public string Text { get; set; }
}..
```

This is a very simple DTO class that has the same properties as the `TodoItem` entity. Now, we are ready to implement our use-cases.

```
 {{if UI=="Blazor"}}

### The Application Service Interface
```

Create a `ITodoAppService` interface under the `Services` folder of the `TodoApp.Contracts` project, as shown below:

```
```csharp
using TodoApp.Services.Dtos;
using Volo.Abp.Application.Services;

namespace TodoApp.Services;

public interface ITodoAppService : IApplicationService
{
 Task<List<TodoItemDto>> GetListAsync();

 Task<TodoItemDto> CreateAsync(string text);

 Task DeleteAsync(Guid id);
}..

 {{end}}
```

### ### The Application Service Implementation

Create a '`TodoAppService`' class under the '`Services`' folder of {{if UI=="Blazor"}}your '`TodoApp.Host`' project{{else}}your project{{end}}, as shown below:

```
```csharp
{{if UI=="Blazor"}}
using TodoApp.Services;
using TodoApp.Services.Dtos;
using TodoApp.Entities;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;
{{else}}
using TodoApp.Entities;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;
{{end}}

namespace TodoApp.Services;

public class TodoAppService : ApplicationService{{if UI=="Blazor"}},
ITodoAppService{{end}}
{
    private readonly IRepository<TodoItem, Guid> _todoItemRepository;

    public TodoAppService(IRepository<TodoItem, Guid> todoItemRepository)
    {
        _todoItemRepository = todoItemRepository;
    }

    // TODO: Implement the methods here...
}
```

```

This class inherits from the '`ApplicationService`' class of the ABP Framework and implements our use-cases. ABP provides default generic `[repositories]`(../../../../../Repositories.md) for the entities. We can use them to perform the fundamental database operations. This class `[injects]`(../../../../../Dependency-Injection.md) '`IRepository<TodoItem, Guid>`', which is the default repository for the '`TodoItem`' entity. We will use it to implement our use cases.

#### #### Getting the Todo Items

Let's start by implementing the '`GetListAsync`' method, which is used to get a list of todo items:

```
```csharp
public async Task<List<TodoItemDto>> GetListAsync()
{
    var items = await _todoItemRepository.GetListAsync();
    return items
        .Select(item => new TodoItemDto
        {
            Id = item.Id,
            Text = item.Text
        }).ToList();
}
```

```

....

We are simply getting the `TodoItem` list from the repository, mapping them to the `TodoItemDto` objects and returning as the result.

#### #### Creating a New Todo Item

The next method is `CreateAsync` and we can implement it as shown below:

```
```csharp
public async Task<TodoItemDto> CreateAsync(string text)
{
    var todoItem = await _todoItemRepository.InsertAsync(
        new TodoItem {Text = text}
    );

    return new TodoItemDto
    {
        Id = todoItem.Id,
        Text = todoItem.Text
    };
}...```

```

The repository's `InsertAsync` method inserts the given `TodoItem` to the database and returns the same `TodoItem` object. It also sets the `Id`, so we can use it on the returning object. We are simply returning a `TodoItemDto` by creating from the new `TodoItem` entity.

Deleting a Todo Item

Finally, we can implement the `DeleteAsync` as the following code block:

```
```csharp
public async Task DeleteAsync(Guid id)
{
 await _todoItemRepository.DeleteAsync(id);
}...```

```

The application service is ready to be used from the UI layer. So, let's implement it.

#### ## User Interface

It is time to show the todo items on the UI! Before starting to write the code, it would be good to remember what we are trying to build. Here's a sample screenshot from the final UI:

```
![todo-list](./todo-list.png)

{{if UI=="MVC"}}

Index.cshtml.cs
```

Open the `Index.cshtml.cs` file in the `Pages` folder and replace the content with the following code block:

```

```csharp
using TodoApp.Services;
using TodoApp.Services.Dtos;
using Volo.Abp.AspNetCore.Mvc.UI.RazorPages;

namespace TodoApp.Pages;

public class IndexModel : AbpPageModel
{
    public List<TodoItemDto> TodoItems { get; set; }

    private readonly TodoAppService _todoAppService;

    public IndexModel(TodoAppService todoAppService)
    {
        _todoAppService = todoAppService;
    }

    public async Task OnGetAsync()
    {
        TodoItems = await _todoAppService.GetListAsync();
    }
}
```

```

This class uses `TodoAppService` to get the list of todo items and assign the `TodoItems` property. We will use it to render the todo items on the razor page.

### ### Index.cshtml

Open the `Index.cshtml` file in the `Pages` folder and replace it with the following content:

```

```xml
@page
@model TodoApp.Pages.IndexModel

@section styles {
    <abp-style src="/Pages/Index.cshtml.css" />
}
@section scripts {
    <abp-script src="/Pages/Index.cshtml.js" />
}



<abp-card>
        <abp-card-header>
            <abp-card-title>
                TODO LIST
            </abp-card-title>
        </abp-card-header>
        <abp-card-body>
            <!-- FORM FOR NEW TODO ITEMS -->
            <form id="NewItemForm" class="row row-cols-lg-auto g-3 align-items-center">
                <div class="col-12">
                    <div class="input-group">


```

```

        <input id="NewItemText" type="text" class="form-control" placeholder="enter text...">
            </div>
        </div>
        <div class="col-12">
            <button type="submit" class="btn btn-primary">Submit</button>
        </div>
    </form>
    <!-- TODO ITEMS LIST -->
    <ul id="TodoList">
        @foreach (var todoItem in Model.TodoItems)
        {
            <li data-id="@todoItem.Id">
                <i class="fa fa-trash-o"></i> @todoItem.Text
            </li>
        }
    </ul>
    </abp-card-body>
</abp-card>
</div>
```

```

We are using ABP's [card tag helper](../../../../UI/AspNetCore/Tag-Helpers/Cards.md) to create a simple card view. You could directly use the standard bootstrap HTML structure, however the ABP [tag helpers](../../../../UI/AspNetCore/Tag-Helpers/Index.md) make it much easier and type safe.

This page imports a CSS and a JavaScript file, so we should also create them.

### ### Index.cshtml.js

Open the `Index.cshtml.js` file in the `Pages` folder and replace with the following content:

```

```js
$(function () {

    // DELETING ITEMS /////////////////////////////////
    $('#TodoList').on('click', 'li i', function(){
        var $li = $(this).parent();
        var id = $li.attr('data-id');

        todoApp.services.todo.delete(id).then(function(){
            $li.remove();
            abp.notify.info('Deleted the todo item.');
        });
    });

    // CREATING NEW ITEMS ///////////////////////////////
    $('#NewItemForm').submit(function(e){
        e.preventDefault();

        var todoText = $('#NewItemText').val();
        todoApp.services.todo.create(todoText).then(function(result){
            $('<li data-id="' + result.id + '">')
                .html('<i class="fa fa-trash-o"></i> ' + result.text)
        });
    });
});
```

```

```

 .appendTo($('#TodoList'));
 $('#NewItemText').val('');
 });
});
}.*;

```

In the first part, we subscribed to the click events of the trash icons near the todo items, deleted the related item on the server and showed a notification on the UI. Also, we removed the deleted item from the DOM, so we wouldn't need to refresh the page.

In the second part, we created a new todo item on the server. If it succeeded, we would then manipulate the DOM to insert a new '`<li>`' element to the todo list. This way, we wouldn't need to refresh the whole page after creating a new todo item.

The interesting part here is how we communicate with the server. See the *\*Dynamic JavaScript Proxies & Auto API Controllers\** section to understand how it works. But now, let's continue and complete the application.

### `### Index.cshtml.css`

As for the final touch, open the '`Index.cshtml.css`' file in the '`Pages`' folder and replace with the following content:

```

```css
#TodoList{
    list-style: none;
    margin: 0;
    padding: 0;
}

#TodoList li {
    padding: 5px;
    margin: 5px 0px;
    border: 1px solid #cccccc;
    background-color: #f5f5f5;
}

#TodoList li i
{
    opacity: 0.5;
}

#TodoList li i:hover
{
    opacity: 1;
    color: #ff0000;
    cursor: pointer;
}
```

```

This is a simple styling for the todo page. We believe that you can do much better :)

Now, you can run the application again and see the result.

### ### Dynamic JavaScript Proxies & Auto API Controllers

In the `Index.cshtml.js` file, we've used the `todoApp.services.todo.delete(...)` and `todoApp.services.todo.create(...)` functions to communicate with the server. These functions are dynamically created by the ABP Framework, thanks to the [Dynamic JavaScript Client Proxy](../../../../UI/AspNetCore/Dynamic-JavaScript-Proxies.md) system. They perform HTTP API calls to the server and return a promise, so you can register a callback to the `then` function as we've done above.

> `services` keyword comes from the namespace (`namespace TodoApp.Services;`). It's a naming convention.

However, you may notice that we haven't created any API Controllers, so how does the server handle these requests? This question brings us to the [Auto API Controller](../../../../API/Auto-API-Controllers.md) feature of the ABP Framework. It automatically converts the application services to \*\*API Controllers\*\* by convention.

If you open [Swagger UI](https://swagger.io/tools/swagger-ui/) by entering the `/swagger` URL in your application, you can see the Todo API:

```
![todo-api](../../todo-api.png)

{{else if UI=="Blazor" || UI=="BlazorServer"}}

Index.razor.cs
```

Open the `Index.razor.cs` file in the `Pages` folder{{if UI=="Blazor"}} in your `Todo.Blazor` project{{end}} and replace the content with the following code block:

```
```csharp
{{if UI=="Blazor"}}
using Microsoft.AspNetCore.Components;
using TodoApp.Services;
using TodoApp.Services.Dtos;
{{else}}
using Microsoft.AspNetCore.Components;
using TodoApp.Services;
using TodoApp.Services.Dtos;
{{end}}

namespace TodoApp.Pages;

public partial class Index
{
    [Inject]
    private {{if UI=="Blazor"}}ITodoAppService{{else}}TodoAppService{{end}}
    TodoAppService { get; set; }

    private List<TodoItemDto> TodoItems { get; set; } = new
    List<TodoItemDto>();
    private string NewTodoText { get; set; }

    protected override async Task OnInitializedAsync()
    {
        TodoItems = await TodoAppService.GetListAsync();
```

```

    }

    private async Task Create()
    {
        var result = await TodoAppService.CreateAsync(NewTodoText);
        TodoItems.Add(result);
        NewTodoText = null;
    }

    private async Task Delete(TodoItemDto todoItem)
    {
        await TodoAppService.DeleteAsync(todoItem.Id);
        await Notify.Info("Deleted the todo item.");
        TodoItems.Remove(todoItem);
    }
}
```

```

This class uses the {{if UI=="Blazor"}}`ITodoAppService`{{else}}`TodoAppService`{{end}} to get the list of todo items. It manipulates the `TodoItems` list after create and delete operations. This way, we don't need to refresh the whole todo list from the server.

### ### Index.razor

Open the `Index.razor` file in the `Pages` folder and replace the content with the following code block:

```

```xml
@page "/"
@inherits TodoAppComponentBase

<div class="container">
    <Card>
        <CardHeader>
            <CardTitle>
                TODO LIST
            </CardTitle>
        </CardHeader>
        <CardBody>
            <!-- FORM FOR NEW TODO ITEMS -->
            <form id="NewItemForm" @onsubmit:preventDefault @onsubmit="() =>
Create()" class="row row-cols-lg-auto g-3 align-items-center">
                <div class="col-12">
                    <div class="input-group">
                        <input name="NewTodoText" type="text" @bind-
value="@NewTodoText" class="form-control" placeholder="enter text..." />
                    </div>
                </div>
                <div class="col-12">
                    <button type="submit" class="btn btn-
primary">Submit</button>
                </div>
            </form>
            <!-- TODO ITEMS LIST -->
            <ul id="TodoList">
                @foreach (var todoItem in TodoItems)

```

```

    {
        <li data-id="@todoItem.Id">
            <i class="far fa-trash-alt"
                @onclick="() => Delete(todoItem)"></i>
            @todoItem.Text
        </li>
    }
</ul>
</CardBody>
</Card>
</div>
```

```

### ### Index.razor.css

As the final touch, open the `Index.razor.css` file in the `Pages` folder and replace it with the following content:

```

```css
#TodoList{
    list-style: none;
    margin: 0;
    padding: 0;
}

#TodoList li {
    padding: 5px;
    margin: 5px 0px;
    border: 1px solid #cccccc;
    background-color: #f5f5f5;
}

#TodoList li i
{
    opacity: 0.5;
}

#TodoList li i:hover
{
    opacity: 1;
    color: #ff0000;
    cursor: pointer;
}
```

```

This is a simple styling for the todo page. We believe that you can do much better :)

Now, you can run the {{if UI=="Blazor"}}`TodoApp.Host` project{{else}}application{{end}} again to see the result.

```
{else if UI=="NG"}
```

### ### Service Proxy Generation

ABP provides a handy feature to automatically create client-side services to easily consume HTTP APIs provided by the server.

You first need to run the `TodoApp` project since the proxy generator reads API definitions from the server application.

Once you run the `TodoApp` project (\*\*Swagger API Definition\*\* will be shown), open a command-line terminal in the directory of `angular` folder and run the following command:

```
```bash
abp generate-proxy -t ng
````
```

If everything goes well, it should generate an output as shown below:

```
```bash
CREATE src/app/proxy/generate-proxy.json (182755 bytes)
CREATE src/app/proxy/README.md (1000 bytes)
CREATE src/app/proxy/services/todo.service.ts (833 bytes)
CREATE src/app/proxy/services/dtos/models.ts (71 bytes)
CREATE src/app/proxy/services/dtos/index.ts (26 bytes)
CREATE src/app/proxy/services/index.ts (81 bytes)
CREATE src/app/proxy/index.ts (61 bytes)
````
```

Then, we can use the `TodoService` to use the server-side HTTP APIs, as we'll do in the next section.

### `home.component.ts`

Open the `/angular/src/app/home/home.component.ts` file and replace its content with the following code block:

```
```ts
import { ToasterService } from "@abp/ng.theme.shared";
import { Component, OnInit } from '@angular/core';
import { TodoItemDto } from "@proxy/services/dtos";
import { TodoService } from "@proxy/services";

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss'],
})

export class HomeComponent implements OnInit {

  todoItems: TodoItemDto[];
  newTodoText: string;

  constructor(
    private todoService: TodoService,
    private toasterService: ToasterService)
  { }

  ngOnInit(): void {
    this.todoService.getList().subscribe(response => {
      this.todoItems = response;
    });
}
````
```

```

create(): void{
 this.todoService.create(this.newTodoText).subscribe((result) => {
 this.todoItems = this.todoItems.concat(result);
 this.newTodoText = null;
 });
}

delete(id: string): void {
 this.todoService.delete(id).subscribe(() => {
 this.todoItems = this.todoItems.filter(item => item.id !== id);
 this.toasterService.info('Deleted the todo item.');
 });
}
```

```

We've used '`TodoService`' to get the list of todo items and assigned the returning value to the '`todoItems`' array. We've also added '`create`' and '`delete`' methods. These methods will be used on the view side.

`### home.component.html`

Open the '`/angular/src/app/home/home.component.html`' file and replace its content with the following code block:

```

```html
<div class="container">
 <div class="card">
 <div class="card-header">
 <div class="card-title">TODO LIST</div>
 </div>
 <div class="card-body">
 <!-- FORM FOR NEW TODO ITEMS -->
 <form class="row row-cols-lg-auto g-3 align-items-center"
 (ngSubmit)="create()">
 <div class="col-12">
 <div class="input-group">
 <input name="NewTodoText" type="text" [(ngModel)]="newTodoText"
 class="form-control" placeholder="enter text..." />
 </div>
 </div>
 <div class="col-12">
 <button type="submit" class="btn btn-primary">Submit</button>
 </div>
 </form>
 <!-- TODO ITEMS LIST -->
 <ul id="TodoList">
 <li *ngFor="let todoItem of todoItems">
 <i class="fa fa-trash-o" (click)="delete(todoItem.id)"></i>
 {{ todoItem.text }}

 </div>
 </div>
```

```

```
### home.component.scss
```

As the final touch, open the `./angular/src/app/home/home.component.scss` file and replace its content with the following code block:

```
```css
#TodoList{
 list-style: none;
 margin: 0;
 padding: 0;
}

#TodoList li {
 padding: 5px;
 margin: 5px 0px;
 border: 1px solid #cccccc;
 background-color: #f5f5f5;
}

#TodoList li i
{
 opacity: 0.5;
}

#TodoList li i:hover
{
 opacity: 1;
 color: #ff0000;
 cursor: pointer;
}
...```

```

This is a simple styling for the todo page. We believe that you can do much better :)

Now, you can run the application again to see the result.

```
 {{end}} ``
```

## ## Conclusion

In this tutorial, we've built a very simple application to warm up with the ABP Framework.

## ## Source Code

You can find the source code of the completed application [here](<https://github.com/abpframework/abp-samples/tree/master/TodoApp-SingleLayer>).

## ## See Also

\* Check the [Web Application Development Tutorial](../../Part-1.md) to see a real-life web application development in a layered architecture using the [Application Startup Template](../../Startup-Templates/Application.md).

## 1.2 With layered architecture

```
Quick Start

```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```

```

This is a single-part quick-start tutorial to build a simple todo application with the ABP Framework. Here's a screenshot from the final application:

![todo-list](todo-list.png)

You can find the source code of the completed application [\[here\]](https://github.com/abpframework/abp-samples/tree/master/TodoApp)(https://github.com/abpframework/abp-samples/tree/master/TodoApp).

This documentation has a video tutorial on \*\*YouTube\*\*!! You can watch it here:

```
 {{if UI=="MVC" && DB == "EF"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/763DV0fwSbk" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

 {{else if UI=="Blazor" && DB=="EF"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/ivxJsi8c7-8" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

 {{else if UI=="BlazorServer" && DB=="EF"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/1BdYg5NLrJs" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

 {{else if UI=="NG" && DB=="EF"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/Lqh1j1H5pkg" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

 {{else if UI=="MVC" && DB=="Mongo"}}

<iframe width="560" height="315" src="https://www.youtube.com/embed/7Rm-
K2re4MI" title="YouTube video player" frameborder="0" allow="accelerometer;
autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture"
allowfullscreen></iframe>
```

```

{{else if UI=="BlazorServer" && DB=="Mongo"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/i23C8hN70As" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

{{else if UI=="Blazor" && DB=="Mongo"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/JpiMiXOBG6A" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

{{else if UI=="NG" && DB=="Mongo"}}

<iframe width="560" height="315"
src="https://www.youtube.com/embed/D0xk9Doxad0" title="YouTube video player"
frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-
media; gyroscope; picture-in-picture" allowfullscreen></iframe>

{{end}}

Pre-Requirements

* An IDE (e.g. \[Visual Studio\](https://visualstudio.microsoft.com/vs/)) that supports \[.NET 7.0+\](https://dotnet.microsoft.com/download/dotnet) development.
* \[Node v16.x\](https://nodejs.org/)

{{if DB=="Mongo"}}

* \[MongoDB Server 4.0+\](https://docs.mongodb.com/manual/administration/install-community)

{{end}}

Install ABP CLI Tool

We will use the \[ABP CLI\](../../CLI.md) to create new ABP solutions. You can run the following command on a terminal window to install this dotnet tool:

```bash
dotnet tool install -g Volo.Abp.Cli
```

Create Your ABP Solution

Create an empty folder, open a command-line terminal and execute the following command in the terminal:

```bash
abp new TodoApp{{if UI=="Blazor"}} -u blazor{{else if UI=="BlazorServer"}} -u
blazor-server{{else if UI=="NG"}} -u angular{{end}}{{if DB=="Mongo"}} -d
mongodb{{end}}
```

```

```
 {{if UI=="NG"}}

This will create a new solution, named *TodoApp* with `angular` and `aspnet-core` folders. Once the solution is ready, open the ASP.NET Core solution in your favorite IDE.
```

```
 {{else}}

This will create a new solution, named *TodoApp*. Once the solution is ready, open it in your favorite IDE.
```

```
 {{end}}

Create the Database
```

```
If you are using Visual Studio, right click on the `TodoApp.DbMigrator` project, select *Set as Startup Project*, then hit *Ctrl+F5* to run it without debugging. It will create the initial database and seed the initial data.
```

```
 {{if DB=="EF"}}

> Some IDEs (e.g. Rider) may have problems for the first run since `DbMigrator` adds the initial migration and re-compiles the project. In this case, open a command-line terminal in the folder of the `TodoApp.DbMigrator` project and execute the `dotnet run` command.
```

```
 {{end}}

Run the Application
```

```
 {{if UI=="MVC" || UI=="BlazorServer"}}

It is good to run the application before starting the development. Ensure the `{{if UI=="BlazorServer"}}` `TodoApp.Blazor` `{{else}}` `TodoApp.Web` `{{end}}` project is the startup project, then run the application (Ctrl+F5 in Visual Studio) to see the initial UI:
```

```
 {{else if UI=="Blazor"}}

It is good to run the application before starting the development. The solution has two main applications;
```

- \* `TodoApp.HttpApi.Host` hosts the server-side HTTP API.
- \* `TodoApp.Blazor` is the client-side Blazor WebAssembly application.

```
Ensure the `TodoApp.HttpApi.Host` project is the startup project, then run the application (Ctrl+F5 in Visual Studio) to see the server-side HTTP API on the [Swagger UI](https://swagger.io/tools/swagger-ui/):
```

```
![todo-swagger-ui-initial](todo-swagger-ui-initial.png)
```

```
You can explore and test your HTTP API with this UI. Now, we can set the `TodoApp.Blazor` as the startup project and run it to open the actual Blazor application UI:
```

```
 {{else if UI=="NG"}}
```

It is good to run the application before starting the development. The solution has two main applications:

- \* `TodoApp.HttpApi.Host` (in the .NET solution) host the server-side HTTP API.
- \* `angular` folder contains the Angular application.

Ensure that the `TodoApp.HttpApi.Host` project is the startup project, then run the application (Ctrl+F5 in Visual Studio) to see the server-side HTTP API on the [Swagger UI](https://swagger.io/tools/swagger-ui/):

```
![todo-swagger-ui-initial](todo-swagger-ui-initial.png)
```

You can explore and test your HTTP API with this UI. If it works, we can run the Angular client application.

You can run the application using the following command:

```
```bash
npm start
````
```

This command takes time, but eventually runs and opens the application in your default browser:

```
 {{end}}
```

```
![todo-ui-initial](todo-ui-initial.png)
```

You can click on the \*Login\* button, use `admin` as the username and `1q2w3E\*` as the password to login to the application.

All ready. We can start coding!

## ## Domain Layer

This application has a single [entity](../../Entities.md) and we'll start by creating it. Create a new `TodoItem` class inside the \*TodoApp.Domain\* project:

```
```csharp
using System;
using Volo.Abp.Domain.Entities;

namespace TodoApp
{
    public class TodoItem : BasicAggregateRoot<Guid>
    {
        public string Text { get; set; }
    }
}
````
```

`BasicAggregateRoot` is the simplest base class to create root entities, and `Guid` is the primary key (`Id`) of the entity here.

## ## Database Integration

```
 {{if DB=="EF"}}
```

Next step is to setup the [Entity Framework Core](../../Entity-Framework-Core.md) configuration.

### ### Mapping Configuration

Open the `TodoAppDbContext` class in the `EntityFrameworkCore` folder of the \*TodoApp.EntityFrameworkCore\* project and add a new `DbSet` property to this class:

```
```csharp
public DbSet<TodoItem> TodoItems { get; set; }
```

Then navigate to the `OnModelCreating` method in the `TodoAppDbContext` class and add the mapping code for the `TodoItem` entity:

```
```csharp
protected override void OnModelCreating(ModelBuilder builder)
{
 base.OnModelCreating(builder);

 /* Include modules to your migration db context */

 builder.ConfigurePermissionManagement();
 ...

 /* Configure your own tables/entities inside here */
 builder.Entity<TodoItem>(b =>
 {
 b.ToTable("TodoItems");
 });
}
...```

```

We've mapped the `TodoItem` entity to the `TodoItems` table in the database.

### ### Code First Migrations

The startup solution is configured to use Entity Framework Core [Code First Migrations](https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations). Since we've changed the database mapping configuration, we should create a new migration and apply changes to the database.

Open a command-line terminal in the directory of the \*TodoApp.EntityFrameworkCore\* project and type the following command:

```
```bash
dotnet ef migrations add Added_TodoItem
```

```

This will add a new migration class to the project:

```
![todo-efcore-migration](todo-efcore-migration.png)
```

You can apply changes to the database using the following command, in the same command-line terminal:

```
```bash
dotnet ef database update
```

> If you are using Visual Studio, you may want to use the `Add-Migration Added_TodoItem` and `Update-Database` commands in the *Package Manager Console (PMC)*. In this case, ensure that {{if UI=="MVC"}} `TodoApp.Web` {{else if UI=="BlazorServer"}} `TodoApp.Blazor` {{else if UI=="Blazor" || UI=="NG"}} `TodoApp.HttpApi.Host` {{end}} is the startup project and `TodoApp.EntityFrameworkCore` is the *Default Project* in PMC.
```

```
 {{else if DB=="Mongo"}}
```

Next step is to setup the [MongoDB](../../MongoDB.md) configuration. Open the `TodoAppMongoDbContext` class in the `MongoDb` folder of the \*TodoApp.MongoDB\* project and make the following changes:

1. Add a new property to the class:

```
```csharp
public IMongoCollection<TodoItem> TodoItems => Collection<TodoItem>();
```

```

2. Add the following code inside the `CreateModel` method:

```
```csharp
modelBuilder.Entity<TodoItem>(b =>
{
    b.CollectionName = "TodoItems";
});
```

{{end}}
```

Now, we can use the ABP repositories to save and retrieve the todo items, as we'll do in the next section.

#### ## Application Layer

An [Application Service](../../Application-Services.md) is used to perform the use cases of the application. We need to perform the following use cases:

- \* Get the list of the todo items
- \* Create a new todo item
- \* Delete an existing todo item

#### ### Application Service Interface

We can start by defining an interface for the application service. Create a new `ITodoAppService` interface in the \*TodoApp.Application.Contracts\* project, as shown below:

```
```csharp
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp.Application.Services;
```

```

namespace TodoApp
{
    public interface ITodoAppService : IApplicationService
    {
        Task<List<TodoItemDto>> GetListAsync();
        Task<TodoItemDto> CreateAsync(string text);
        Task DeleteAsync(Guid id);
    }
}
```
Data Transfer Object

`GetListAsync` and `CreateAsync` methods return `TodoItemDto`. `ApplicationService` typically gets and returns DTOs ([Data Transfer Objects](../../Data-Transfer-Objects.md)) instead of entities. So, we should define the DTO class here. Create a new `TodoItemDto` class inside the *TodoApp.Application.Contracts* project:
```

```

```csharp
using System;

namespace TodoApp
{
    public class TodoItemDto
    {
        public Guid Id { get; set; }
        public string Text { get; set; }
    }
}
```

```

This is a very simple DTO class that matches our `TodoItem` entity. We are ready to implement the `ITodoAppService`.

### Application Service Implementation

Create a `TodoAppService` class inside the \*TodoApp.Application\* project, as shown below:

```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace TodoApp
{
    public class TodoAppService : ApplicationService, ITodoAppService
    {
        private readonly IRepository<TodoItem, Guid> _todoItemRepository;

        public TodoAppService(IRepository<TodoItem, Guid> todoItemRepository)
        {
            _todoItemRepository = todoItemRepository;
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                _todoItemRepository?.Dispose();
            }
            base.Dispose(disposing);
        }
    }
}
```

```

```

 }

 // TODO: Implement the methods here...
 }
}
```

```

This class inherits from the `ApplicationService` class of the ABP Framework and implements the `ITodoAppService` that was defined before. ABP provides default generic [repositories](../../Repositories.md) for the entities. We can use them to perform the fundamental database operations. This class [injects](../../Dependency-Injection.md) `IRepository<TodoItem, Guid>`, which is the default repository for the `TodoItem` entity. We will use it to implement the use cases described before.

Getting Todo Items

Let's start by implementing the `GetListAsync` method:

```

```csharp
public async Task<List<TodoItemDto>> GetListAsync()
{
 var items = await _todoItemRepository.GetListAsync();
 return items
 .Select(item => new TodoItemDto
 {
 Id = item.Id,
 Text = item.Text
 }).ToList();
}
```

```

We are simply getting the complete `TodoItem` list from the database, mapping them to `TodoItemDto` objects and returning as the result.

Creating a New Todo Item

Next method is `CreateAsync` and we can implement it as shown below:

```

```csharp
public async Task<TodoItemDto> CreateAsync(string text)
{
 var todoItem = await _todoItemRepository.InsertAsync(
 new TodoItem {Text = text}
);

 return new TodoItemDto
 {
 Id = todoItem.Id,
 Text = todoItem.Text
 };
}
```

```

The repository's `InsertAsync` method inserts the given `TodoItem` to the database and returns the same `TodoItem` object. It also sets the `Id`, so we can use it on the returning object. We are simply returning a `TodoItemDto` by creating from the new `TodoItem` entity.

Deleting a Todo Item

Finally, we can implement the `DeleteAsync` as the following code block:

```
```csharp
public async Task DeleteAsync(Guid id)
{
 await _todoItemRepository.DeleteAsync(id);
}
...```

```

The application service is ready to be used from the UI layer.

#### ## User Interface Layer

It is time to show the todo items on the UI! Before starting to write the code, it would be good to remember what we are trying to build. Here's a sample screenshot from the final UI:

```
![todo-list](todo-list.png)

> **We will keep the UI side minimal for this tutorial to make the tutorial simple and focused. See the [web application development tutorial](../Part-1.md) to build real-life pages with all aspects.**

{{if UI=="MVC"}}

Index.cshtml.cs
```

Open the `Index.cshtml.cs` file in the `Pages` folder of the \*TodoApp.Web\* project and replace the content with the following code block:

```
```csharp
using System.Collections.Generic;
using System.Threading.Tasks;

namespace TodoApp.Web.Pages
{
    public class IndexModel : TodoAppPageModel
    {
        public List<TodoItemDto> TodoItems { get; set; }

        private readonly ITodoAppService _todoAppService;

        public IndexModel(ITodoAppService todoAppService)
        {
            _todoAppService = todoAppService;
        }

        public async Task OnGetAsync()
        {
            TodoItems = await _todoAppService.GetListAsync();
        }
    }
}
...```

```

This class uses the `ITodoAppService` to get the list of todo items and assign the `TodoItems` property. We will use it to render the todo items on the razor page.

Index.cshtml

Open the `Index.cshtml` file in the `Pages` folder of the *TodoApp.Web* project and replace it with the following content:

```
```xml
@page
@model TodoApp.Web.Pages.IndexModel
@section styles {
 <abp-style src="/Pages/Index.css" />
}
@section scripts {
 <abp-script src="/Pages/Index.js" />
}
<div class="container">
 <abp-card>
 <abp-card-header>
 <abp-card-title>
 TODO LIST
 </abp-card-title>
 </abp-card-header>
 <abp-card-body>
 <!-- FORM FOR NEW TODO ITEMS -->
 <form id="NewItemForm" class="row row-cols-lg-auto g-3 align-items-center">
 <div class="col-12">
 <div class="input-group">
 <input id="NewItemText" type="text" class="form-control" placeholder="enter text...">
 </div>
 </div>
 <div class="col-12">
 <button type="submit" class="btn btn-primary">Submit</button>
 </div>
 </form>
 <!-- TODO ITEMS LIST -->
 <ul id="TodoList">
 @foreach (var todoItem in Model.TodoItems)
 {
 <li data-id="@todoItem.Id">
 <i class="fa fa-trash-o"></i> @todoItem.Text

 }

 </abp-card-body>
 </abp-card>
</div>
````
```

We are using ABP's [card tag helper](../../UI/AspNetCore/Tag-Helpers/Cards.md) to create a simple card view. You could directly use the standard bootstrap HTML structure, however the ABP [tag

`helpers]`(`../UI/AspNetCore/Tag-Helpers/Index.md`) make it much easier and type safe.

This page imports a CSS and a JavaScript file, so we should also create them.

Index.js

Open the `'Index.js'` file in the `'Pages'` folder of the `*TodoApp.Web*` project and replace it with the following content:

```
```js
$(function () {

 // DELETING ITEMS /////////////////////////////////
 $('#TodoList').on('click', 'li i', function(){
 var $li = $(this).parent();
 var id = $li.attr('data-id');

 todoApp.todo.delete(id).then(function(){
 $li.remove();
 abp.notify.info('Deleted the todo item.');
 });
 });

 // CREATING NEW ITEMS ///////////////////////////////
 $('#NewItemForm').submit(function(e){
 e.preventDefault();

 var todoText = $('#NewItemText').val();
 todoApp.todo.create(todoText).then(function(result){
 $('- ' +
 ' ' + result.text)
 .appendTo($('#TodoList'));
 $('#NewItemText').val('');
 });
 });
});
```

```

In the first part, we are subscribing to the click events of the trash icons near the todo items, deleting the related item on the server and showing a notification on the UI. Also, we are removing the deleted item from the DOM, so we don't need to refresh the page.

In the second part, we are creating a new todo item on the server. If it succeeds, we are then manipulating the DOM to insert a new `''` element to the todo list. This way we don't need to refresh the whole page after creating a new todo item.

The interesting part here is how we communicate with the server. See the `*Dynamic JavaScript Proxies & Auto API Controllers*` section to understand how it works. But now, let's continue and complete the application.

Index.css

As the final touch, open the `'Index.css'` file in the `'Pages'` folder of the `*TodoApp.Web*` project and replace it with the following content:

```
```css
#TodoList{
 list-style: none;
 margin: 0;
 padding: 0;
}

#TodoList li {
 padding: 5px;
 margin: 5px 0px;
 border: 1px solid #cccccc;
 background-color: #f5f5f5;
}

#TodoList li i
{
 opacity: 0.5;
}

#TodoList li i:hover
{
 opacity: 1;
 color: #ff0000;
 cursor: pointer;
}
```

```

This is a simple styling for the todo page. We believe that you can do much better :)

Now, you can run the application again and see the result.

Dynamic JavaScript Proxies & Auto API Controllers

In the `Index.js` file, we've used the `todoApp.todo.delete(...)` and `todoApp.todo.create(...)` functions to communicate with the server. These functions are dynamically created by the ABP Framework, thanks to the [Dynamic JavaScript Client Proxy](../../UI/AspNetCore/Dynamic-JavaScript-Proxies.md) system. They perform HTTP API calls to the server and return a promise, so you can register a callback to the `then` function as we've done above.

However, you may notice that we haven't created any API Controllers, so how does the server handle these requests? This question brings us to the [Auto API Controller](../../API/Auto-API-Controllers.md) feature of the ABP Framework. It automatically converts the application services to API Controllers by convention.

If you open the [Swagger UI](<https://swagger.io/tools/swagger-ui/>) by entering the `/swagger` URL in your application, you can see the Todo API:

```
![todo-api](todo-api.png)

{{else if UI=="Blazor" || UI=="BlazorServer"}}

### Index.razor.cs
```

Open the `'Index.razor.cs'` file in the `'Pages'` folder of the `*TodoApp.Blazor*` project and replace the content with the following code block:

```
```csharp
using Microsoft.AspNetCore.Components;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace TodoApp.Blazor.Pages
{
 public partial class Index
 {
 [Inject]
 private ITodoAppService TodoAppService { get; set; }

 private List<TodoItemDto> TodoItems { get; set; } = new
List<TodoItemDto>();
 private string NewTodoText { get; set; }

 protected override async Task OnInitializedAsync()
 {
 TodoItems = await TodoAppService.GetListAsync();
 }

 private async Task Create()
 {
 var result = await TodoAppService.CreateAsync(NewTodoText);
 TodoItems.Add(result);
 NewTodoText = null;
 }

 private async Task Delete(TodoItemDto todoItem)
 {
 await TodoAppService.DeleteAsync(todoItem.Id);
 await Notify.Info("Deleted the todo item.");
 TodoItems.Remove(todoItem);
 }
 }
}...```

```

This class uses `'ITodoAppService'` to perform operations for the todo items. It manipulates the `'TodoItems'` list after create and delete operations. This way, we don't need to refresh the whole todo list from the server.

```
 {{if UI=="Blazor"}}


```

See the `*Dynamic C# Proxies & Auto API Controllers*` section below to learn how we could inject and use the application service interface from the Blazor application which is running on the browser! But now, let's continue and complete the application.

```
 {{end # Blazor}}
```

```
Index.razor
```

Open the `'Index.razor'` file in the `'Pages'` folder of the `*TodoApp.Blazor*` project and replace the content with the following code block:

```

```xml
@page "/"
@inherits TodoAppComponentBase
<div class="container">
    <Card>
        <CardHeader>
            <CardTitle>
                TODO LIST
            </CardTitle>
        </CardHeader>
        <CardBody>
            <!-- FORM FOR NEW TODO ITEMS -->
            <form id="NewItemForm" @onsubmit:preventDefault @onsubmit="() =>
Create()" class="row row-cols-lg-auto g-3 align-items-center">
                <div class="col-12">
                    <div class="input-group">
                        <input name="NewTodoText" type="text" @bind-
value="@NewTodoText" class="form-control" placeholder="enter text..." />
                    </div>
                </div>
                <div class="col-12">
                    <button type="submit" class="btn btn-
primary">Submit</button>
                </div>
            </form>
            <!-- TODO ITEMS LIST -->
            <ul id="TodoList">
                @foreach (var todoItem in TodoItems)
                {
                    <li data-id="@todoItem.Id">
                        <i class="far fa-trash-alt"
                            @onclick="() => Delete(todoItem)"></i> @todoItem.Text
                    </li>
                }
            </ul>
        </CardBody>
    </Card>
</div>
```

```

### ### Index.razor.css

As the final touch, open the `Index.razor.css` file in the `Pages` folder of the \*TodoApp.Blazor\* project and replace it with the following content:

```

```css
#TodoList{
    list-style: none;
    margin: 0;
    padding: 0;
}

#TodoList li {
    padding: 5px;
    margin: 5px 0px;
    border: 1px solid #cccccc;
}
```

```

```
 background-color: #f5f5f5;
 }

#TodoList li i
{
 opacity: 0.5;
}

#TodoList li i:hover
{
 opacity: 1;
 color: #ff0000;
 cursor: pointer;
}
...
```

This is a simple styling for the todo page. We believe that you can do much better :)

Now, you can run the application again to see the result.

```
{{if UI=="Blazor"}}

Dynamic C# Proxies & Auto API Controllers
```

In the `Index.razor.cs` file, we've injected (with the `[Inject]` attribute) and used the `ITodoAppService` just like using a local service. Remember that the Blazor application is running on the browser while the implementation of this application service is running on the server.

The magic is done by the ABP Framework's [Dynamic C# Client Proxy](../../API/Dynamic-CSharp-API-Clients.md) system. It uses the standard `HttpClient` and performs HTTP API requests to the remote server. It also handles all the standard tasks for us, including authorization, JSON serialization and exception handling.

However, you may ask that we haven't created any API Controller, so how does the server handle these requests? This question brings us to the [Auto API Controller](../../API/Auto-API-Controllers.md) feature of the ABP Framework. It automatically converts the application services to API Controllers by convention.

If you run the `TodoApp.HttpApi.Host` application, you can see the Todo API:

```
![todo-api](todo-api.png)

{{end # Blazor}}

{{else if UI=="NG"}}

Service Proxy Generation
```

ABP provides a handy feature to automatically create client-side services to easily consume HTTP APIs provided by the server.

You first need to run the `TodoApp.HttpApi.Host` project since the proxy generator reads API definitions from the server application.

> **Warning**: There is an issue with IIS Express: it doesn't allow connecting to the application from another process. If you are using Visual Studio, select the '`TodoApp.HttpApi.Host`' instead of IIS Express in the run button drop-down list, as shown in the figure below:

![run-without-iisexpress](run-without-iisexpress.png)

Once you run the '`TodoApp.HttpApi.Host`' project, open a command-line terminal in the '`angular`' folder and type the following command:

```
```bash
abp generate-proxy -t ng
````
```

If everything goes well, it should generate an output as shown below:

```
```bash
CREATE src/app/proxy/generate-proxy.json (170978 bytes)
CREATE src/app/proxy/README.md (1000 bytes)
CREATE src/app/proxy/todo.service.ts (794 bytes)
CREATE src/app/proxy/models.ts (66 bytes)
CREATE src/app/proxy/index.ts (58 bytes)
````
```

We can then use '`todoService`' to use the server-side HTTP APIs, as we'll do in the next section.

### ### `home.component.ts`

Open the '`/angular/src/app/home/home.component.ts`' file and replace its content with the following code block:

```
```js
import { ToasterService } from '@abp/ng.theme.shared';
import { Component, OnInit } from '@angular/core';
import { TodoItemDto, TodoService } from '@proxy';

@Component({
  selector: 'app-home',
  templateUrl: './home.component.html',
  styleUrls: ['./home.component.scss']
})
export class HomeComponent implements OnInit {

  todoItems: TodoItemDto[];
  newTodoText: string;

  constructor(
    private todoService: TodoService,
    private toasterService: ToasterService)
  {}

  ngOnInit(): void {
    this.todoService.getList().subscribe(response => {
      this.todoItems = response;
    });
  }
}
```

```

    create(): void{
      this.todoService.create(this.newTodoText).subscribe((result) => {
        this.todoItems = this.todoItems.concat(result);
        this.newTodoText = null;
      });
    }

    delete(id: string): void {
      this.todoService.delete(id).subscribe(() => {
        this.todoItems = this.todoItems.filter(item => item.id !== id);
        this.toasterService.info('Deleted the todo item.');
      });
    }
  }
}

...

```

We've used `todoService` to get the list of todo items and assigned the returning value to the `todoItems` array. We've also added `create` and `delete` methods. These methods will be used on the view side.

home.component.html

Open the `/angular/src/app/home/home.component.html` file and replace its content with the following code block:

```

```html
<div class="container">
 <div class="card">
 <div class="card-header">
 <div class="card-title">TODO LIST</div>
 </div>
 <div class="card-body">
 <!-- FORM FOR NEW TODO ITEMS -->
 <form class="row row-cols-lg-auto g-3 align-items-center"
 (ngSubmit)="create()">
 <div class="col-12">
 <div class="input-group">
 <input name="NewTodoText" type="text" [(ngModel)]="newTodoText"
 class="form-control" placeholder="enter text..." />
 </div>
 </div>
 <div class="col-12">
 <button type="submit" class="btn btn-primary">Submit</button>
 </div>
 </form>
 <!-- TODO ITEMS LIST -->
 <ul id="TodoList">
 <li *ngFor="let todoItem of todoItems">
 <i class="fa fa-trash-o" (click)="delete(todoItem.id)"></i>
 {{ todoItem.text }}

 </div>
 </div>
</div>
```

```

home.component.scss

As the final touch, open the `/angular/src/app/home/home.component.scss` file and replace its content with the following code block:

```
```css
#TodoList{
 list-style: none;
 margin: 0;
 padding: 0;
}

#TodoList li {
 padding: 5px;
 margin: 5px 0px;
 border: 1px solid #cccccc;
 background-color: #f5f5f5;
}

#TodoList li i
{
 opacity: 0.5;
}

#TodoList li i:hover
{
 opacity: 1;
 color: #ff0000;
 cursor: pointer;
}
````
```

This is a simple styling for the todo page. We believe that you can do much better :)

Now, you can run the application again to see the result.

}}}

Conclusion

In this tutorial, we've built a very simple application to warm up for the ABP Framework. If you are looking to build a serious application, please check the [web application development tutorial](../Part-1.md) which covers all the aspects of real-life web application development.

Source Code

You can find source code of the completed application [here](<https://github.com/abpframework/abp-samples/tree/master/TodoApp>).

See Also

* [Web Application Development Tutorial](../Part-1.md)

2 Getting Started

```
# Getting Started: Overall  
## Select the Solution Architecture
```

This tutorial has multiple versions. Please select the one that fits you the best:

- * **[Single-Layer Solution](Getting-Started-Single-Layered.md)**: Creates a single-project solution. Recommended for building an application with a **simpler and easy to understand** architecture.
- * **[Layered Solution Architecture](Getting-Started.md)**: A fully layered (multiple projects) solution based on [Domain Driven Design](Domain-Driven-Design.md) practices. Recommended for long-term projects that need a **maintainable and extensible** codebase.

2.1 Web Application- Layered Architecture

```
# Getting Started  
```json  
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"],
 "Tiered": ["Yes", "No"]
}
...

> This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and **{{ DB_Value }}** as the database provider. For other options, please change the preference on top of this document.
```

This tutorial explains how to \*\*create and run\*\* a new web application using the ABP Framework. Follow the steps below;

1. [Setup your development environment](Getting-Started-Setup-Environment.md)
2. [Creating a new solution](Getting-Started-Create-Solution.md)
3. [Running the solution](Getting-Started-Running-Solution.md)

#### 2.1.1 1: Setup Your Development Environment

```
Getting Started
```json  
//[doc-params]  
{  
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],  
    "DB": ["EF", "Mongo"],  
    "Tiered": ["Yes", "No"]  
}
```

....

> This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and **{{ DB_Value }}** as the database provider. For other options, please change the preference on top of this document.

Setup Your Development Environment

First things first! Let's setup your development environment before creating the project.

Pre-Requirements

The following tools should be installed on your development machine:

* An IDE (e.g. [Visual Studio](https://visualstudio.microsoft.com/vs/)) that supports [.NET 7.0+](https://dotnet.microsoft.com/download/dotnet) development.

 {{ if UI != "Blazor" }}

 * [Node v16 or v18](https://nodejs.org/)

 * [Yarn v1.20+ (not v2)](https://classic.yarnpkg.com/en/docs/install) ^{[1] (#f-yarn)} or npm v6+ (already installed with Node)

 {{ end }}

 {{ if Tiered == "Yes" }}

 * [Redis](https://redis.io/) (the startup solution uses the Redis as the [distributed cache](Caching.md)).

 {{ end }}

 {{ if UI != "Blazor" }}

 ^{1} _Yarn v2 works differently and is not supported._ ^{[
](#a-yarn)}

 {{ end }}

Install the ABP CLI

[ABP CLI](./CLI.md) is a command line interface that is used to automate some common tasks for ABP based solutions. First, you need to install the ABP CLI using the following command:

```
```shell
dotnet tool install -g Volo.Abp.Cli
```
```

If you've already installed, you can update it using the following command:

```
```shell
dotnet tool update -g Volo.Abp.Cli
```
```

Next Step

* [Creating a new solution](Getting-Started-Create-Solution.md)

2.1.2 2: Creating a New Solution

```
# Getting Started

```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"],
 "Tiered": ["Yes", "No"]
}
```

```

> This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and **{{ DB_Value }}** as the database provider. For other options, please change the preference on top of this document.

Create a New Project

We will use the ABP CLI to create a new ABP project.

> Alternatively, you can **create and download** projects from the [ABP Framework website](<https://abp.io/get-started>) by easily selecting all options from the page.

Use the `'new'` command of the ABP CLI to create a new project:

```
```shell
abp new Acme.BookStore{{if UI == "NG"}} -u angular{{else if UI == "Blazor"}} -u blazor{{else if UI == "BlazorServer"}} -u blazor-server{{end}}{{if DB == "Mongo"}} -d mongodb{{end}}{{if Tiered == "Yes"}}{{if UI == "MVC" || UI == "BlazorServer"}} --tiered{{else}} --separate-auth-server{{end}}{{end}}
```

```

You can use different level of namespaces; e.g. BookStore, Acme.BookStore or Acme.Retail.BookStore.

```
 {{ if Tiered == "Yes" }}
```

```
 {{ if UI == "MVC" || UI == "BlazorServer" }}
```

** `--tiered` argument is used to create N-tiered solution where authentication server, UI and API layers are physically separated.*

```
 {{ else }}
```

** `--separate-auth-server` argument is used to separate the Auth Server application from the API host application. If not specified, you will have a single endpoint on the server.*

```
 {{ end }}
```

```
 {{ end }}
```

> [ABP CLI document](./CLI.md) covers all of the available commands and options.

Mobile Development

If you want to include a [React Native](https://reactnative.dev/) project in your solution, add `'-m react-native'` (or ` '--mobile react-native'`) argument to project creation command. This is a basic React Native startup template to develop mobile applications integrated to your ABP based backends.

See the [Getting Started with the React Native](Getting-Started-React-Native.md) document to learn how to configure and run the React Native application.

The Solution Structure

The solution has a layered structure (based on the [Domain Driven Design](Domain-Driven-Design.md)) and contains unit & integration test projects. See the [application template document](Startup-Templates/Application.md) to understand the solution structure in details.

```
 {{ if DB == "Mongo" }}
```

MongoDB Transactions

The [startup template](Startup-templates/Index.md) **disables** transactions in the `.MongoDB` project by default. If your MongoDB server supports transactions, you can enable it in the *YourProjectMongoDbModule* class's `ConfigureServices` method:

```
```csharp
Configure<AbpUnitOfWorkDefaultOptions>(options =>
{
 options.TransactionBehavior = UnitOfWorkTransactionBehavior.Auto;
});...
```

> Or you can delete that code since `Auto` is already the default behavior.

```
 {{ end }}
```

## ## Next Step

\* [Running the solution](Getting-Started-Running-Solution.md)

### 2.1.3 3: Running the Solution

#### # Getting Started

```
```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"],
    "Tiered": ["Yes", "No"]
}...
```

> This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and **{{ DB_Value }}** as the database provider. For other options, please change the preference on top of this document.

```

## Create the Database

### Connection String

Check the **connection string** in the `appsettings.json` file under the {{if Tiered == "Yes"}}`.AuthServer` and `.HttpApi.Host` projects{{else}}{{if UI=="MVC"}}`.Web` project{{else if UI=="BlazorServer"}}`.Blazor` project{{else}}`HttpApi.Host` project{{end}}{{end}}.

{{ if DB == "EF" }}

```json
"ConnectionStrings": {
 "Default": "Server=(LocalDb)\MSSQLLocalDB;Database=BookStore;Trusted_Connection=True"
}
```

> **About the Connection Strings and Database Management Systems**
>
> The solution is configured to use **Entity Framework Core** with **MS SQL Server** by default. However, if you've selected another DBMS using the `--dbms` parameter on the ABP CLI `new` command (like `--dbms MySQL`), the connection string might be different for you.
>
> EF Core supports [various](https://docs.microsoft.com/en-us/ef/core/providers/) database providers and you can use any supported DBMS. See [the Entity Framework integration document](Entity-Framework-Core.md) to learn how to [switch to another DBMS](Entity-Framework-Core-Other-DBMS.md) if you need later.

### Database Migrations

The solution uses the [Entity Framework Core Code First Migrations](https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/?tabs=dotnet-core-cli). It comes with a `DbMigrator` console application which **applies the migrations** and also **seeds the initial data**. It is useful on **development** as well as on **production** environment.

> `DbMigrator` project has its own `appsettings.json`. So, if you have changed the connection string above, you should also change this one.

### The Initial Migration

`DbMigrator` application automatically **creates the Initial migration** on first run.

**If you are using Visual Studio, you can skip to the *Running the DbMigrator* section.** However, other IDEs (e.g. Rider) may have problems for the first run since it adds the initial migration and compiles the project. In this case, open a command line terminal in the folder of the `DbMigrator` project and run the following command:

```bash
dotnet run
```

```

For the next time, you can just run it in your IDE as you normally do.

Running the DbMigrator

Right click to the `DbMigrator` project and select **Set as StartUp Project**

![set-as-startup-project](images/set-as-startup-project.png)

Hit F5 (or Ctrl+F5) to run the application. It will have an output like shown below:

![db-migrator-output](images/db-migrator-output.png)

> Initial [seed data](Data-Seeding.md) creates the `admin` user in the database (with the password is `1q2w3E*`) which is then used to login to the application. So, you need to use `DbMigrator` at least once for a new database.

```
{} else if DB == "Mongo" {}
```

```
```json
"ConnectionStrings": {
 "Default": "mongodb://localhost:27017/BookStore"
}
```

```

The solution is configured to use **MongoDB** in your local computer, so you need to have a MongoDB server instance up and running or change the connection string to another MongoDB server.

Seed Initial Data

The solution comes with a `DbMigrator` console application which **seeds the initial data**. It is useful on **development** as well as on **production** environment.

> `DbMigrator` project has its own `appsettings.json`. So, if you have changed the connection string above, you should also change this one.

Right click to the `DbMigrator` project and select **Set as StartUp Project**

![set-as-startup-project](images/set-as-startup-project.png)

Hit F5 (or Ctrl+F5) to run the application. It will have an output like shown below:

![db-migrator-output](images/db-migrator-output.png)

> Initial [seed data](Data-Seeding.md) creates the `admin` user in the database (with the password is `1q2w3E*`) which is then used to login to the application. So, you need to use `DbMigrator` at least once for a new database.

```
{} end {}
```

Run the Application

```

{{ if UI == "MVC" || UI == "BlazorServer" }}

> Before starting the application, run `abp install-libs` command in your Web
directory to restore the client-side libraries. This will populate the `libs`
folder.

{{ if Tiered == "Yes" }}

> Tiered solutions use **Redis** as the distributed cache. Ensure that it is
installed and running in your local computer. If you are using a remote Redis
Server, set the configuration in the `appsettings.json` files of the projects
below.

1. Ensure that the `'.AuthServer'` project is the startup project. Run this
application that will open a **login** page in your browser.

> Use Ctrl+F5 in Visual Studio (instead of F5) to run the application without
debugging. If you don't have a debug purpose, this will be faster.

You can login, but you cannot enter to the main application here. This is
**just the authentication server**.

2. Ensure that the `'.HttpApi.Host'` project is the startup project and run the
application which will open a **Swagger UI** in your browser.

![swagger-ui](images/swagger-ui.png)

This is the HTTP API that is used by the web application.

3. Lastly, ensure that the `{{if UI=="MVC"}}` `'.Web'` `{{else}}` `'.Blazor'` `{{end}}` project
is the startup project and run the application which will open a
**welcome** page in your browser

![mvc-tiered-app-home](images/bookstore-home-2.png)

Click to the **login** button which will redirect you to the *authentication
server* to login to the application:

![bookstore-login](images/bookstore-login-2.png)

{{ else # Tiered != "Yes" }}

Ensure that the `{{if UI=="MVC"}}` `'.Web'` `{{else}}` `'.Blazor'` `{{end}}` project is the
startup project. Run the application which will open the **login** page in
your browser:

> Use Ctrl+F5 in Visual Studio (instead of F5) to run the application without
debugging. If you don't have a debug purpose, this will be faster.

![bookstore-login](images/bookstore-login-2.png)

{{ end # Tiered }}

{{ else # UI != MVC || BlazorServer }}

### Running the HTTP API Host (Server Side)

```

```
 {{ if Tiered == "Yes" }}
```

> Tiered solutions use Redis as the distributed cache. Ensure that it is installed and running in your local computer. If you are using a remote Redis Server, set the configuration in the `appsettings.json` files of the projects below.

Ensure that the `AuthServer` project is the startup project. Run the application which will open a **login** page in your browser.

> Use Ctrl+F5 in Visual Studio (instead of F5) to run the application without debugging. If you don't have a debug purpose, this will be faster.

You can login, but you cannot enter to the main application here. This is **just the authentication server**.

Ensure that the `HttpApi.Host` project is the startup project and run the application which will open a Swagger UI:

```
 {{ else # Tiered == "No" }}
```

Ensure that the `HttpApi.Host` project is the startup project and run the application which will open a Swagger UI:

> Use Ctrl+F5 in Visual Studio (instead of F5) to run the application without debugging. If you don't have a debug purpose, this will be faster.

```
 {{ end # Tiered }}
```

![swagger-ui](images/swagger-ui.png)

You can see the application APIs and test them here. Get [more info](https://swagger.io/tools/swagger-ui/) about the Swagger UI.

```
 {{ end # UI }}
```

```
 {{ if UI == "Blazor" }}
```

Running the Blazor Application (Client Side)

Go to the Blazor project folder, open a command line terminal, type the `abp bundle -f` command (If the project was created by ABP Cli tool, you don't need to do this).

Ensure that the `Blazor` project is the startup project and run the application.

> Use Ctrl+F5 in Visual Studio (instead of F5) to run the application without debugging. If you don't have a debug purpose, this will be faster.

Once the application starts, click to the **Login** link on to header, which redirects you to the authentication server to enter a username and password:

![bookstore-login](images/bookstore-login-2.png)

```
 {{ else if UI == "NG" }}
```

Running the Angular Application (Client Side)

Go to the `angular` folder, open a command line terminal, type the `yarn` command (we suggest to the [yarn](<https://yarnpkg.com/>) package manager while `npm install` will also work)

```
```bash
yarn
````
```

Once all node modules are loaded, execute `yarn start` (or `npm start`) command:

```
```bash
yarn start
````
```

It may take a longer time for the first build. Once it finishes, it opens the Angular UI in your default browser with the [localhost:4200](<http://localhost:4200/>) address.

![bookstore-login](images/bookstore-login-2.png)

{} end {}

Enter **admin** as the username and **1q2w3E*** as the password to login to the application. The application is up and running. You can start developing your application based on this startup template.

See Also

- * [Web Application Development Tutorial](Tutorials/Part-1.md)
- * [Application Startup Template](Startup-Templates/Application.md)

2.2 Web Application- Single-Layered Architecture

Getting Started

```
```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}
````
```

> This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and **{{ DB_Value }}** as the database provider. For other options, please change the preference on top of this document.

This tutorial explains how to **create and run** a new Single-Layered web application using the ABP Framework. Follow the steps below:

1. [Setup your development environment](Getting-Started-Setup-Environment-Single-Layer.md)
2. [Creating a new solution](Getting-Started-Create-Solution-Single-Layer.md)
3. [Running the solution](Getting-Started-Running-Solution-Single-Layer.md)

2.2.1 1: Setup Your Development Environment

```
# Getting Started

```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}
```

> This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and **{{ DB_Value }}** as the database provider. For other options, please change the preference on top of this document.
```

Setup Your Development Environment

First things first! Let's setup your development environment before creating the project.

Pre-Requirements

The following tools should be installed on your development machine:

- * An IDE (e.g. [Visual Studio](https://visualstudio.microsoft.com/vs/)) (<https://visualstudio.microsoft.com/vs/>) that supports [.NET 7.0+](https://dotnet.microsoft.com/download/dotnet) (<https://dotnet.microsoft.com/download/dotnet>) development.

```
    {{ if UI != "Blazor" }}
* [Node v16 or v18] (https://nodejs.org/)
* [Yarn v1.20+ (not v2)] (https://classic.yarnpkg.com/en/docs/install) <sup id="a-yarn">[1]</sup> or npm v6+ (already installed with Node)
    {{ end }}
```

```
    {{ if UI != "Blazor" }}
```

*^{1} _Yarn v2 works differently and is not supported._ ^[
#a-yarn]*

```
    {{ end }}
```

Install the ABP CLI

[\[ABP CLI\]](#) ([./CLI.md](#)) is a command line interface that is used to automate some common tasks for ABP based solutions. First, you need to install the ABP CLI using the following command:

```
```shell
dotnet tool install -g Volo.Abp.Cli
```
```

If you've already installed, you can update it using the following command:

```
```shell
dotnet tool update -g Volo.Abp.Cli
```
```

....

Next Step

- * [Creating a new solution](Getting-Started-Create-Solution-Single-Layer.md)

2.2.2 2: Creating a New Solution

Getting Started

```
```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}
...```

```

> This document assumes that you prefer to use \*\*{{ UI\_Value }}\*\* as the UI framework and \*\*{{ DB\_Value }}\*\* as the database provider. For other options, please change the preference on top of this document.

#### ## Create a New Project

We will use the ABP CLI to create a new ABP project.

> You can also use the ABP CLI Command Generator on the [ABP Framework website](https://abp.io/get-started) by easily selecting all options from the page.

Use the `new` command of the ABP CLI to create a new project:

```
```shell
abp new Acme.BookStore -t app-nolayers{{if UI == "NG"}} -u angular{{else if
UI == "Blazor"}} -u blazor{{else if UI == "BlazorServer"}} -u blazor-
server{{end}}{{if DB == "Mongo"}} -d mongodb{{end}}
```

```

\*You can use different level of namespaces; e.g. BookStore, Acme.BookStore or Acme.Retail.BookStore.\*

> [ABP CLI document](./CLI.md) covers all of the available commands and options.

#### ## The Solution Structure

The solution structure is based on the [Single-Layer Startup Template](Startup-Templates/Application-Single-Layer.md) where everything is in one project instead of the [Domain Driven Design](Domain-Driven-Design.md). You can check its [documentation](Startup-Templates/Application-Single-Layer.md) for more details.

```
 {{ if DB == "Mongo" }}```

```

#### ## MongoDB Transactions

The [\[startup template\]](#)(Startup-templates/Index.md) \*\*disables\*\* transactions in the `'.MongoDB'` project by default. If your MongoDB server supports transactions, you can enable it in the `*YourProjectModule*` class's `'ConfigureMongoDB'` method:

```
```csharp
Configure<AbpUnitOfWorkDefaultOptions>(options =>
{
    options.TransactionBehavior = UnitOfWorkTransactionBehavior.Enabled;
//or UnitOfWorkTransactionBehavior.Auto
});...
```

> Or you can delete that code since `'Auto'` is already the default behavior.

```
{} end {}
```

Next Step

* [\[Running the solution\]](#)(Getting-Started-Running-Solution-Single-Layer.md)

2.2.3 3: Running the Solution

Getting Started

```
```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}...
```

> This document assumes that you prefer to use `**{{ UI_Value }}**` as the UI framework and `**{{ DB_Value }}**` as the database provider. For other options, please change the preference on top of this document.

#### ## Create the Database

#### ### Connection String

Check the `**connection string**` in the `'appsettings.json'` file under the `'YourProject'` project.

```
{} if DB == "EF" {}
```

```
```json
"ConnectionStrings": {
    "Default": 
"Server=(LocalDb)\MSSQLLocalDB;Database=BookStore;Trusted_Connection=True"
}...
```

> **About the Connection Strings and Database Management Systems**

>

> The solution is configured to use `**Entity Framework Core**` with `**MS SQL Server**` by default. However, if you've selected another DBMS using the `'-`

```
dbms` parameter on the ABP CLI `new` command (like `--dbms MySQL`), the connection string might be different for you.  
> EF Core supports [various](https://docs.microsoft.com/en-us/ef/core/providers/) database providers and you can use any supported DBMS. See [the Entity Framework integration document](Entity-Framework-Core.md) to learn how to [switch to another DBMS](Entity-Framework-Core-Other-DBMS.md) if you need later.  
{{ else if DB == "Mongo" }}  
```json  
"ConnectionStrings": {
 "Default": "mongodb://localhost:27017/BookStore"
}
...
The solution is configured to use **MongoDB** in your local computer, so you need to have a MongoDB server instance up and running or change the connection string to another MongoDB server.
```

```
{{ end }}
Seed Initial Data
```

Before running the application, you need to create the database and seed the initial data. To do that, you can run the following command in the directory of your project (in the same folder of the `'.csproj'` file):

```
```bash  
dotnet run --migrate-database  
```
```

```
Run the Application
{{if UI=="MVC" || UI=="BlazorServer"}}
```
```

Running the application is pretty straight-forward, you can run the application with any IDE that supports .NET or by running the `dotnet run` CLI command in the directory of your project:

```
 {{else if UI=="Blazor"}}  
```  
Running the application is pretty straight-forward, you just need to run the `TodoApp.Host` application with any IDE that supports .NET or by running the `dotnet run` CLI command in the directory of your project.
```

> \*\*Note:\*\* The `host` application hosts and serves the `blazor` application. Therefore, you should run the `host` application only.

After the application runs, open the application in your default browser.

```
 {{else if UI=="NG"}}
```

The solution has two main applications:

```
* `TodoApp` (in the .NET solution) hosts the server-side HTTP API, so the Angular application can consume it. (server-side application)
* `angular` folder contains the Angular application. (client-side application)
```

Firstly, run the `TodoApp` project in your favorite IDE (or run the `dotnet run` CLI command on your project directory) to see the server-side HTTP API on [Swagger UI](https://swagger.io/tools/swagger-ui/).

![swagger-ui](images/swagger-ui.png)

You can explore and test your HTTP API with this UI. If it works, then we can run the Angular client application.

You can run the application using the following (or `yarn start`) command:

```
```bash
npm start
````
```

This command takes time, but eventually runs and opens the application in your default browser.

}}}

After running the project, the index page should be seen as below:

![single-layer-index-page](images/single-layer-index-page.png)

Enter \*\*admin\*\* as the username and \*\*1q2w3E\*\*\* as the password to login to the application. The application is up and running. You can start developing your application based on this startup template.

![bookstore-login-2](images/bookstore-login-2.png)

## 2.3 Console Application

```
Console Application Startup Template
```

This template is used to create a minimalist console application project.

### ## How to Start With?

First, install the [ABP CLI](./CLI.md) if you haven't installed before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
````
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.MyConsoleApp -t console
````
```

`Acme.MyConsoleApp` is the solution name, like \*YourCompany.YourProduct\*. You can use single level, two-levels or three-levels naming.

## ## Solution Structure

After you use the above command to create a solution, you will have a solution like shown below:

![basic-console-application-solution](./images/basic-console-application-solution.png)

\* `HelloWorldService` is a sample service that implements the `ITransientDependency` interface to register this service to the [dependency injection](./Dependency-Injection.md) system.

## 2.4 WPF Application

### # WPF Application Startup Template

This template is used to create a minimalist WPF application project.

#### ## How to Start With?

First, install the [ABP CLI](./CLI.md) if you haven't installed before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
````
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.MyWpfApp -t wpf
````
```

`Acme.MyWpfApp` is the solution name, like \*YourCompany.YourProduct\*. You can use single level, two-levels or three-levels naming.

## ## Solution Structure

After you use the above command to create a solution, you will have a solution like shown below:

![basic-wpf-application-solution](./images/basic-wpf-application-solution.png)

\* `HelloWorldService` is a sample service that implements the `ITransientDependency` interface to register this service to the [dependency injection](./Dependency-Injection.md) system.

## 2.5 MAUI

### # MAUI Application Startup Template

This template is used to create a minimalist MAUI application project.

#### ## How to Start With?

First, install the [ABP CLI](../CLI.md) if you haven't installed before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
````
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.MyMauiApp -t maui
````
```

`Acme.MyMauiApp` is the solution name, like *\*YourCompany.YourProduct\**. You can use single level, two-levels or three-levels naming.

## ## Solution Structure

After you use the above command to create a solution, you will have a solution like shown below:

![basic-maui-application-solution](../images/basic-maui-application-solution.png)  
\* `HelloWorldService` is a sample service that implements the `ITransientDependency` interface to register this service to the [dependency injection](./Dependency-Injection.md) system.

## 2.6 Empty Web Project

### # Getting Started with an Empty ASP.NET Core MVC / Razor Pages Application

This tutorial explains how to start ABP from scratch with minimal dependencies. You generally want to start with the \*\*[startup template](Getting-Started-AspNetCore-MVC-Template.md)\*\*.

#### ## Create a New Project

1. Create a new AspNet Core Web Application with Visual Studio 2022 (17.0.0+):



2. Configure your new project:



3. Press the create button:

![create-aspnet-core-application](images/create-aspnet-core-application.png)

#### ## Install Volo.Abp.AspNetCore.Mvc Package

You can use the [ABP CLI](CLI.md) to install the Volo.Abp.AspNetCore.Mvc package to your project. Execute the following command in the folder of the .csproj file that you want to install the package on:

```
```bash
abp add-package Volo.Abp.AspNetCore.Mvc
```
```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [[the package description page](#)](<https://abp.io/package-detail/Volo.Abp.AspNetCore.Mvc>).

### ## Create the First ABP Module

ABP is a modular framework and it requires a \*\*startup (root) module\*\* class derived from ``AbpModule``:

```
```C#
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.Hosting;
using Volo.Abp;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.Modularity;

namespace BasicAspNetCoreApplication
{
    [DependsOn(typeof(AbpAspNetCoreMvcModule))]
    public class AppModule : AbpModule
    {
        public override void
OnApplicationInitialization(ApplicationInitializationContext context)
        {
            var app = context.GetApplicationBuilder();
            var env = context.GetEnvironment();

            // Configure the HTTP request pipeline.
            if (env.IsDevelopment())
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change
this for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();
            app.UseRouting();
            app.UseConfiguredEndpoints();
        }
    }
}
```
```

``AppModule`` is a good name for the startup module for an application.

ABP packages define module classes and a module can depend on another. In the code above, the ``AppModule`` depends on the ``AbpAspNetCoreMvcModule`` (defined by the [\[Volo.Abp.AspNetCore.Mvc\]](#)(<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc>) package). It's common to add a ``DependsOn`` attribute after installing a new ABP NuGet package.

Instead of the `Startup` class, we are configuring an ASP.NET Core pipeline in this module class.

## ## The Program Class

Next step is to modify the `Program` class to integrate to the ABP module system:

```
```C#
using BasicAspNetCoreApplication;

var builder = WebApplication.CreateBuilder(args);

await builder.AddApplicationAsync<AppModule>();

var app = builder.Build();

await app.InitializeApplicationAsync();
await app.RunAsync();
```

``builder.AddApplicationAsync<AppModule>();`` adds all services defined in all modules starting from the ``AppModule``.

``app.InitializeApplicationAsync()`` initializes and starts the application.
```

## ## Run the Application!

That's all! Run the application, it will just work as expected.

## ## Using Autofac as the Dependency Injection Framework

While ASP.NET Core's Dependency Injection (DI) system is fine for basic requirements, [\[Autofac\]](https://autofac.org/)(<https://autofac.org/>) provides advanced features like Property Injection and Method Interception which are required by ABP to perform advanced application framework features.

Replacing ASP.NET Core's DI system by Autofac and integrating to ABP is pretty easy.

### 1. Install

[[Volo.Abp.Autofac](https://www.nuget.org/packages/Volo.Abp.Autofac)](<https://www.nuget.org/packages/Volo.Abp.Autofac>) package

....

`Install-Package Volo.Abp.Autofac`

....

### 2. Add the ``AbpAutofacModule`` Dependency

```
```C#
[DependsOn(typeof(AbpAspNetCoreMvcModule))]
[DependsOn(typeof(AbpAutofacModule))] //Add dependency to ABP Autofac module
public class AppModule : AbpModule
{
    ...
}
```
```

3. Update `Program.cs` to use Autofac:

```
```C#
using BasicAspNetCoreApplication;

var builder = WebApplication.CreateBuilder(args);

builder.Host.UseAutofac(); //Add this line

await builder.AddApplicationAsync<AppModule>();

var app = builder.Build();

await app.InitializeApplicationAsync();
await app.RunAsync();
````
```

## Source Code

Get source code of the sample project created in this tutorial from [\[here\]](https://github.com/abpframework/abp-samples/tree/master/BasicAspNetCoreApplication)(<https://github.com/abpframework/abp-samples/tree/master/BasicAspNetCoreApplication>).

## 3 Tutorials

### 3.1 Web Application Development

#### 3.1.1 1: Creating the Server Side

```
Web Application Development Tutorial - Part 1: Creating the Server Side
```json
// [doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
````
```

## About This Tutorial

In this tutorial series, you will build an ABP based web application named `Acme.BookStore`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* \*\*{{DB\_Value}}\*\* as the database provider.
- \* \*\*{{UI\_Value}}\*\* as the UI Framework.

This tutorial is organized as the following parts:

- \*\*Part 1: Creating the server side (this part)\*\*
- [Part 2: The book list page](Part-2.md)
- [Part 3: Creating, updating and deleting books](Part-3.md)
- [Part 4: Integration tests](Part-4.md)
- [Part 5: Authorization](Part-5.md)
- [Part 6: Authors: Domain layer](Part-6.md)
- [Part 7: Authors: Database Integration](Part-7.md)
- [Part 8: Authors: Application Layer](Part-8.md)

- [Part 9: Authors: User Interface](Part-9.md)
- [Part 10: Book to Author Relation](Part-10.md)

### ### Download the Source Code

This tutorial has multiple versions based on your \*\*UI\*\* and \*\*Database\*\* preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [MVC (Razor Pages) UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- \* [Blazor UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* [Angular UI with MongoDB](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [this guide](..../KB/Windows-Path-Too-Long-Fix.md).

```
 {{if UI == "MVC" && DB == "EF"}}
```

### ### Video Tutorial

This part is also recorded as a video tutorial and \*\*<a href="https://www.youtube.com/watch?v=cJzyIFFAlp8&list=PLsNcLT2aHJcPNaCf7Io3D bMN6yAk\_DgWJ&index=1" target="\_blank">published on YouTube</a>\*\*.

```
 {{end}}
```

## ## Creating the Solution

Before starting the development, create a new solution named `Acme.BookStore` and run it by following the [getting started tutorial](..../Getting-Started.md).

### ## Create the Book Entity

\*\*Domain layer\*\* in the startup template is separated into two projects:

- `Acme.BookStore.Domain` contains your [entities](..../Entities.md), [domain services](..../Domain-Services.md) and other core domain objects.
- `Acme.BookStore.Domain.Shared` contains `constants`, `enums` or other domain related objects that can be shared with clients.

So, define your entities in the domain layer (`Acme.BookStore.Domain` project) of the solution.

The main entity of the application is the `Book`. Create a `Books` folder (namespace) in the `Acme.BookStore.Domain` project and add a `Book` class inside it:

```
```csharp
using System;
using Volo.Abp.Domain.Entities.Auditing;

namespace Acme.BookStore.Books;

public class Book : AuditedAggregateRoot<Guid>
```

```

{
    public string Name { get; set; }

    public BookType Type { get; set; }

    public DateTime PublishDate { get; set; }

    public float Price { get; set; }
}...

```

* ABP Framework has two fundamental base classes for entities: `AggregateRoot` and `Entity`. **Aggregate Root** is a [Domain Driven Design](../Domain-Driven-Design.md) concept which can be thought as a root entity that is directly queried and worked on (see the [entities document](../Entities.md) for more).

- * The `Book` entity inherits from the `AuditedAggregateRoot` which adds some base [auditing](../Audit-Logging.md) properties (like `CreationTime`, `CreatorId`, `LastModificationTime`...) on top of the `AggregateRoot` class. ABP automatically manages these properties for you.
- * `Guid` is the **primary key type** of the `Book` entity.

> This tutorial leaves the entity properties with **public get/set** for the sake of simplicity. See the [entities document](../Entities.md) if you want to learn more about DDD best practices.

BookType Enum

The `Book` entity uses the `BookType` enum. Create a `Books` folder (namespace) in the `Acme.BookStore.Domain.Shared` project and add a `BookType` inside it:

```
```csharp
namespace Acme.BookStore.Books;

public enum BookType
{
 Undefined,
 Adventure,
 Biography,
 Dystopia,
 Fantastic,
 Horror,
 Science,
 ScienceFiction,
 Poetry
}...
```

```

The final folder/file structure should be as shown below:

![bookstore-book-and-booktype](images/bookstore-book-and-booktype.png)

Add the Book Entity to the DbContext

```
{{if DB == "EF"}}

```

EF Core requires that you relate the entities with your `DbContext`. The easiest way to do so is adding a `DbSet` property to the `BookStoreDbContext` class in the `Acme.BookStore.EntityFrameworkCore` project, as shown below:

```
```csharp
public class BookStoreDbContext : AbpDbContext<BookStoreDbContext>
{
 public DbSet<Book> Books { get; set; }
 //...
}
```
{{end}}
```

```
 {{if DB == "Mongo"}}
Add a `IMongoCollection<Book> Books` property to the
```

`BookStoreMongoDbContext` inside the `Acme.BookStore.MongoDB` project:

```
```csharp
public class BookStoreMongoDbContext : AbpMongoDbContext
{
 public IMongoCollection<Book> Books => Collection<Book>();
 //...
}
```

```

```
 {{end}}
```

```
 {{if DB == "EF"}}
### Map the Book Entity to a Database Table
```

Navigate to the `OnModelCreating` method in the `BookStoreDbContext` class and add the mapping code for the `Book` entity:

```
```csharp
using Acme.BookStore.Books;
...

namespace Acme.BookStore.EntityFrameworkCore;

public class BookStoreDbContext :
 AbpDbContext<BookStoreDbContext>,
 IIdentityDbContext,
 ITenantManagementDbContext
{
 ...

 protected override void OnModelCreating(ModelBuilder builder)
 {
 base.OnModelCreating(builder);

 /* Include modules to your migration db context */

 builder.ConfigurePermissionManagement();
 ...
 }
}
```

```

/* Configure your own tables/entities inside here */

builder.Entity<Book>(b =>
{
 b.ToTable(BookStoreConsts.DbTablePrefix + "Books",
 BookStoreConsts.DbSchema);
 b.ConfigureByConvention(); //auto configure for the base class
props
 b.Property(x => x.Name).IsRequired().HasMaxLength(128);
});
}
...

```

\* `BookStoreConsts` has constant values for the schema and table prefixes for your tables. You don't have to use it, but it's suggested to control the table prefixes in a single point.

\* The `ConfigureByConvention()` method gracefully configures/maps the inherited properties. Always use it for all your entities.

### ### Add Database Migration

The startup solution is configured to use [Entity Framework Core Code First Migrations](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). Since we've changed the database mapping configuration, we should create a new migration and apply changes to the database.

Open a command-line terminal in the directory of the `Acme.BookStore.EntityFrameworkCore` project and type the following command:

```
```bash
dotnet ef migrations add Created_Book_Entity
```

```

This will add a new migration class to the project:

```
![bookstore-efcore-migration](./images/bookstore-efcore-migration.png)

> If you are using Visual Studio, you may want to use the `Add-Migration Created_Book_Entity` and `Update-Database` commands in the *Package Manager Console (PMC)*. In this case, ensure that `Acme.BookStore.EntityFrameworkCore` is the startup project in Visual Studio and `Acme.BookStore.EntityFrameworkCore` is the *Default Project* in PMC.
```

}}{end}}

### ### Add Sample Seed Data

> It's good to have some initial data in the database before running the application. This section introduces the [Data Seeding](../Data-Seeding.md) system of the ABP framework. You can skip this section if you don't want to create the data seeding, but it is suggested to follow along and learn this useful ABP Framework feature.

Create a class deriving from the `IDataSeedContributor` in the `\*.Domain` project by copying the following code:

```
```csharp
```

```

using System;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore;

public class BookStoreDataSeederContributor
    : IDataSeedContributor, ITransientDependency
{
    private readonly IRepository<Book, Guid> _bookRepository;

    public BookStoreDataSeederContributor(IRepository<Book, Guid> bookRepository)
    {
        _bookRepository = bookRepository;
    }

    public async Task SeedAsync(DataSeedContext context)
    {
        if (await _bookRepository.GetCountAsync() <= 0)
        {
            await _bookRepository.InsertAsync(
                new Book
                {
                    Name = "1984",
                    Type = BookType.Dystopia,
                    PublishDate = new DateTime(1949, 6, 8),
                    Price = 19.84f
                },
                autoSave: true
            );

            await _bookRepository.InsertAsync(
                new Book
                {
                    Name = "The Hitchhiker's Guide to the Galaxy",
                    Type = BookType.ScienceFiction,
                    PublishDate = new DateTime(1995, 9, 27),
                    Price = 42.0f
                },
                autoSave: true
            );
        }
    }
}

* This code simply uses the `IRepository<Book, Guid>` (the default [repository](../../Repositories.md)) to insert two books to the database in case there weren't any books in it.

### Update the Database

Run the `Acme.BookStore.DbMigrator` application to update the database:

```

```
![bookstore-db migrator-on-solution](images/bookstore-db migrator-on-solution.png)

`.DbMigrator` is a console application that can be run to **migrate the database schema** and **seed the data** on **development** and **production** environments.
```

Create the Application Service

The application layer is separated into two projects:

- * `Acme.BookStore.Application.Contracts` contains your [DTO](../Data-Transfer-Objects.md)s and [application service](../Application-Services.md) interfaces.
- * `Acme.BookStore.Application` contains the implementations of your application services.

In this section, you will create an application service to get, create, update and delete books using the `CrudAppService` base class of the ABP Framework.

BookDto

``CrudAppService`` base class requires to define the fundamental DTOs for the entity. Create a `Books` folder (namespace) in the `Acme.BookStore.Application.Contracts` project and add a `BookDto` class inside it:

```
```csharp
using System;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Books;

public class BookDto : AuditedEntityDto<Guid>
{
 public string Name { get; set; }

 public BookType Type { get; set; }

 public DateTime PublishDate { get; set; }

 public float Price { get; set; }
}
...```

```

\* \*\*DTO\*\* classes are used to \*\*transfer data\*\* between the \*presentation layer\* and the \*application layer\*. See the [Data Transfer Objects document](https://docs.abp.io/en/abp/latest/Data-Transfer-Objects) for more details.

- \* The `BookDto` is used to transfer the book data to the presentation layer in order to show the book information on the UI.
- \* The `BookDto` is derived from the `AuditedEntityDto<Guid>` which has audit properties just like the `Book` entity defined above.

It will be needed to map the `Book` entities to the `BookDto` objects while returning books to the presentation layer.

[AutoMapper](https://automapper.org) library can automate this conversion

when you define the proper mapping. The startup template comes with AutoMapper pre-configured. So, you can just define the mapping in the `BookStoreApplicationAutoMapperProfile` class in the `Acme.BookStore.Application` project:

```
```csharp
using Acme.BookStore.Books;
using AutoMapper;

namespace Acme.BookStore;

public class BookStoreApplicationAutoMapperProfile : Profile
{
    public BookStoreApplicationAutoMapperProfile()
    {
        CreateMap<Book, BookDto>();
    }
}...```

```

> See the [object to object mapping](../Object-To-Object-Mapping.md) document for details.

CreateUpdateBookDto

Create a `CreateUpdateBookDto` class in the `Books` folder (namespace) of the `Acme.BookStore.Application.Contracts` project:

```
```csharp
using System;
using System.ComponentModel.DataAnnotations;

namespace Acme.BookStore.Books;

public class CreateUpdateBookDto
{
 [Required]
 [StringLength(128)]
 public string Name { get; set; }

 [Required]
 public BookType Type { get; set; } = BookType.Undefined;

 [Required]
 [DataType(DataType.Date)]
 public DateTime PublishDate { get; set; } = DateTime.Now;

 [Required]
 public float Price { get; set; }
}...```

```

\* This `DTO` class is used to get a book information from the user interface while creating or updating the book.

\* It defines data annotation attributes (like `[Required]`) to define validations for the properties. `DTO`'s are [automatically validated](<https://docs.abp.io/en/abp/latest/Validation>) by the ABP framework.

As done to the `BookDto` above, we should define the mapping from the `CreateUpdateBookDto` object to the `Book` entity. The final class will be as shown below:

```
```csharp
using Acme.BookStore.Books;
using AutoMapper;

namespace Acme.BookStore;

public class BookStoreApplicationAutoMapperProfile : Profile
{
    public BookStoreApplicationAutoMapperProfile()
    {
        CreateMap<Book, BookDto>();
        CreateMap<CreateUpdateBookDto, Book>();
    }
}...
```

```

### ### IBookAppService

Next step is to define an interface for the application service. Create an `IBookAppService` interface in the `Books` folder (namespace) of the `Acme.BookStore.Application.Contracts` project:

```
```csharp
using System;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;

namespace Acme.BookStore.Books;

public interface IBookAppService :
    ICrudAppService< //Defines CRUD methods
        BookDto, //Used to show books
        Guid, //Primary key of the book entity
        PagedAndSortedResultRequestDto, //Used for paging/sorting
        CreateUpdateBookDto> //Used to create/update a book
{
}
```

```

\* Defining interfaces for the application services \*\*are not required\*\* by the framework. However, it's suggested as a best practice.  
\* `ICrudAppService` defines common \*\*CRUD\*\* methods: `GetAsync`, `GetListAsync`, `CreateAsync`, `UpdateAsync` and `DeleteAsync`. It's not required to extend it. Instead, you could inherit from the empty `IApplicationService` interface and define your own methods manually (which will be done for the authors in the next parts).  
\* There are some variations of the `ICrudAppService` where you can use separated DTOs for each method (like using different DTOs for create and update).

### ### BookAppService

It is time to implement the `IBookAppService` interface. Create a new class, named `BookAppService` in the `Books` namespace (folder) of the `Acme.BookStore.Application` project:

```
```csharp
using System;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Books;

public class BookAppService :
    CrudAppService<
        Book, //The Book entity
        BookDto, //Used to show books
        Guid, //Primary key of the book entity
        PagedAndSortedResultRequestDto, //Used for paging/sorting
        CreateUpdateBookDto>, //Used to create/update a book
    IBookAppService //implement the IBookAppService
{
    public BookAppService(IRepository<Book, Guid> repository)
        : base(repository)
    {

    }
}
```
...
```

- \* `BookAppService` is derived from `CrudAppService<...>` which implements all the CRUD (create, read, update, delete) methods defined by the `ICrudAppService`.
- \* `BookAppService` injects ` IRepository<Book, Guid>` which is the default repository for the `Book` entity. ABP automatically creates default repositories for each aggregate root (or entity). See the [repository document](<https://docs.abp.io/en/abp/latest/Repositories>).
- \* `BookAppService` uses `IObjectMapper` service ([see]([../Object-To-Object-Mapping.md](#))) to map the `Book` objects to the `BookDto` objects and `CreateUpdateBookDto` objects to the `Book` objects. The Startup template uses the [AutoMapper](<http://automapper.org/>) library as the object mapping provider. We have defined the mappings before, so it will work as expected.

### ## Auto API Controllers

In a typical ASP.NET Core application, you create \*\*API Controllers\*\* to expose the application services as \*\*HTTP API\*\* endpoints. This allows browsers or 3rd-party clients to call them over HTTP.

ABP can [\*\*automagically\*\*]([../API/Auto-API-Controllers.md](#)) configure your application services as MVC API Controllers by convention.

### ### Swagger UI

The startup template is configured to run the [Swagger UI](<https://swagger.io/tools/swagger-ui/>) using the [Swashbuckle.AspNetCore](<https://github.com/domaindrivendev/Swashbuckle.AspNetCore>) library. Run the application {{if UI=="MVC"}}`Acme.BookStore.Web`{{else}}`Acme.BookStore.HttpApi.Host`{{end}}

by pressing `CTRL+F5` and navigate to `https://localhost:<port>/swagger/` on your browser. Replace `**<port>**` with your own port number.

You will see some built-in service endpoints as well as the `Book` service and its REST-style endpoints:

```
![bookstore-swagger](./images/bookstore-swagger.png)
```

Swagger has a nice interface to test the APIs.

If you try to execute the `[GET] /api/app/book` API to get a list of books, the server returns such a JSON result:

```
```json
{
  "totalCount": 2,
  "items": [
    {
      "name": "The Hitchhiker's Guide to the Galaxy",
      "type": 7,
      "publishDate": "1995-09-27T00:00:00",
      "price": 42,
      "lastModificationTime": null,
      "lastModifierId": null,
      "creationTime": "2020-07-03T21:04:18.4607218",
      "creatorId": null,
      "id": "86100bb6-cbc1-25be-6643-39f62806969c"
    },
    {
      "name": "1984",
      "type": 3,
      "publishDate": "1949-06-08T00:00:00",
      "price": 19.84,
      "lastModificationTime": null,
      "lastModifierId": null,
      "creationTime": "2020-07-03T21:04:18.3174016",
      "creatorId": null,
      "id": "41055277-cce8-37d7-bb37-39f62806960b"
    }
  ]
}...
```

```

That's pretty cool since we haven't written a single line of code to create the API controller, but now we have a fully working REST API!

## ## The Next Part

See the [next part](Part-2.md) of this tutorial.

### 3.1.2 2: The Book List Page

```
Web Application Development Tutorial - Part 2: The Book List Page
```json
//[doc-params]
{
```

```

    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```
About This Tutorial
```

In this tutorial series, you will build an ABP based web application named '`Acme.BookStore`'. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* `DB_Value` as the ORM provider.
- \* `UI_Value` as the UI Framework.

This tutorial is organized as the following parts:

- [Part 1: Creating the server side](Part-1.md)
- \*\*Part 2: The book list page (this part)\*\*
- [Part 3: Creating, updating and deleting books](Part-3.md)
- [Part 4: Integration tests](Part-4.md)
- [Part 5: Authorization](Part-5.md)
- [Part 6: Authors: Domain layer](Part-6.md)
- [Part 7: Authors: Database Integration](Part-7.md)
- [Part 8: Authors: Application Layer](Part-8.md)
- [Part 9: Authors: User Interface](Part-9.md)
- [Part 10: Book to Author Relation](Part-10.md)

### ### Download the Source Code

This tutorial has multiple versions based on your **UI** and **Database** preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [MVC (Razor Pages) UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- \* [Blazor UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* [Angular UI with MongoDB](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [this guide](..../KB/Windows-Path-Too-Long-Fix.md).

```
{{if UI == "MVC" && DB == "EF"}}
```

### ### Video Tutorial

This part is also recorded as a video tutorial and [published on YouTube](https://www.youtube.com/watch?v=UDNLLiPiBiw&list=PLsNcLT2aHJcPNaCf7Io3D bMN6yAk_DgWJ&index=2).

```
{{end}}
```

```
{{if UI == "MVC"}}
```

### ## Dynamic JavaScript Proxies

It's common to call the HTTP API endpoints via AJAX from the \*\*JavaScript\*\* side. You can use `$.ajax` or another tool to call the endpoints. However, ABP offers a better way.

ABP \*\*dynamically\*\* creates \*\*[JavaScript Proxies](../UI/AspNetCore/Dynamic-JavaScript-Proxies.md)\*\* for all the API endpoints. So, you can use any \*\*endpoint\*\* just like calling a \*\*JavaScript function\*\*.

### ### Testing in the Developer Console

You can easily test the JavaScript proxies using your favorite browser's \*\*Developer Console\*\*. Run the application, open your browser's \*\*developer tools\*\* (\*shortcut is generally F12\*), switch to the \*\*Console\*\* tab, type the following code and press enter:

```
```js
acme.bookStore.books.book.getList({}).done(function (result)
{ console.log(result); });

* `acme.bookStore.books` is the namespace of the `BookAppService` converted to [camelCase](https://en.wikipedia.org/wiki/Camel_case).
* `book` is the conventional name for the `BookAppService` (removed `AppService` postfix and converted to camelCase).
* `getList` is the conventional name for the `GetListAsync` method defined in the `CrudAppService` base class (removed `Async` postfix and converted to camelCase).
* The `{}` argument is used to send an empty object to the `GetListAsync` method which normally expects an object of type `PagedAndSortedResultRequestDto` that is used to send paging and sorting options to the server (all properties are optional with default values, so you can send an empty object).
* The `getList` function returns a `promise`. You can pass a callback to the `then` (or `done`) function to get the result returned from the server.
```

Running this code produces the following output:

```
![bookstore-javascript-proxy-console](images/bookstore-javascript-proxy-console.png)
```

You can see the **book list** returned from the server. You can also check the **network** tab of the developer tools to see the client to server communication:

```
![bookstore-getlist-result-network](images/bookstore-getlist-result-network.png)
```

Let's **create a new book** using the `create` function:

```
```js
acme.bookStore.books.book.create({
 name: 'Foundation',
 type: 7,
 publishDate: '1951-05-24',
 price: 21.5
}).then(function (result) {
 console.log('successfully created the book with id: ' + result.id);
});
```

....

> If you downloaded the source code of the tutorial and are following the steps from the sample, you should also pass the `authorId` parameter to the create method for \*\*creating a new book\*\*.

You should see a message in the console that looks something like this:

```
```text
successfully created the book with id: 439b0ea8-923e-8e1e-5d97-39f2c7ac4246
````
```

Check the `Books` table in the database to see the new book row. You can try `get`, `update` and `delete` functions yourself.

We will use these dynamic proxy functions in the next sections to communicate with the server.

```
{{end}}
```

#### ## Localization

Before starting the UI development, we first want to prepare the localization texts (you normally do this when needed while developing your application).

Localization texts are located under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project:

```
![bookstore-localization-files](images/bookstore-localization-files-v2.png)
```

Open the `en.json` (\*the English translations\*) file and change the content as shown below:

```
```json
{
  "Culture": "en",
  "Texts": {
    "Menu:Home": "Home",
    "Welcome": "Welcome",
    "LongWelcomeMessage": "Welcome to the application. This is a startup project based on the ABP framework. For more information, visit abp.io.",
    "Menu:BookStore": "Book Store",
    "Menu:Books": "Books",
    "Actions": "Actions",
    "Close": "Close",
    "Delete": "Delete",
    "Edit": "Edit",
    "PublishDate": "Publish date",
    "NewBook": "New book",
    "Name": "Name",
    "Type": "Type",
    "Price": "Price",
    "CreationTime": "Creation time",
    "AreYouSure": "Are you sure?",
    "AreYouSureToDelete": "Are you sure you want to delete this item?",
    "Enum:BookType.0": "Undefined",
    "Enum:BookType.1": "Adventure",
    "Enum:BookType.2": "Biography",
    "Enum:BookType.3": "Science Fiction"
  }
}
```

```

    "Enum:BookType.3": "Dystopia",
    "Enum:BookType.4": "Fantastic",
    "Enum:BookType.5": "Horror",
    "Enum:BookType.6": "Science",
    "Enum:BookType.7": "Science fiction",
    "Enum:BookType.8": "Poetry"
}
}...

```

* Localization key names are arbitrary. You can set any name. We prefer some conventions for specific text types;

- * Add `Menu:` prefix for menu items.
- * Use `Enum:<enum-type>.<enum-value>` or `<enum-type>.<enum-value>` naming convention to localize the enum members. When you do it like that, ABP can automatically localize the enums in some proper cases.

If a text is not defined in the localization file, it **falls back** to the localization key (as ASP.NET Core's standard behavior).

> ABP's localization system is built on the [ASP.NET Core's standard localization](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization>) system and extends it in many ways. Check the [localization document](./Localization.md) for details.

```

{{if UI == "MVC"}}
## Create a Books Page

```

It's time to create something visible and usable! Instead of the classic MVC, we will use the [Razor Pages UI](<https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start>) approach which is recommended by Microsoft.

Create a `Books` folder under the `Pages` folder of the `Acme.BookStore.Web` project. Add a new Razor Page by right clicking the Books folder then selecting **Add > Razor Page** menu item. Name it as `Index`:

![bookstore-add-index-page](images/bookstore-add-index-page-v2.png)

Open the `Index.cshtml` and change the whole content as shown below:

```

```html
@page
@using Acme.BookStore.Web.Pages.Books
@model IndexModel

<h2>Books</h2>
```

```

`Index.cshtml.cs` content should be like that:

```

```csharp
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Acme.BookStore.Web.Pages.Books;

public class IndexModel : PageModel

```

```

{
 public void OnGet()
 {

 }
}..

```

#### Add Books Page to the Main Menu

Open the `BookStoreMenuContributor` class in the `Menus` folder and add the following code to the end of the `ConfigureMainMenuAsync` method:

```
```csharp
context.Menu.AddItem(
    new ApplicationMenuItem(
        "BooksStore",
        l["Menu:BookStore"],
        icon: "fa fa-book"
    ).AddItem(
        new ApplicationMenuItem(
            "BooksStore.Books",
            l["Menu:Books"],
            url: "/Books"
        )
    )
);..
```

Run the project, login to the application with the username `admin` and the password `1q2w3E*` and you can see that the new menu item has been added to the main menu:

![bookstore-menu-items](images/bookstore-new-menu-item-2.png)

When you click on the Books menu item under the Book Store parent, you will be redirected to the new empty Books Page.

Book List

We will use the [Datatables.net](<https://datatables.net/>) jQuery library to show the book list. Datatables library completely works via AJAX, it is fast, popular and provides a good user experience.

> Datatables library is configured in the startup template, so you can directly use it in any page without including any style or script file for your page.

Index.cshtml

Change the `Pages/Books/Index.cshtml` as the following:

```
```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@model IndexModel
```

```

@inject IStringLocalizer<BookStoreResource> L
@section scripts
{
 <abp-script src="/Pages/Books/Index.js" />
}
<abp-card>
 <abp-card-header>
 <h2>@L["Books"]</h2>
 </abp-card-header>
 <abp-card-body>
 <abp-table striped-rows="true" id="BooksTable"></abp-table>
 </abp-card-body>
</abp-card>
```
* ``abp-script` [tag helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro) is used to add external **scripts** to the page. It has many additional features compared to the standard `script` tag. It handles **minification** and **versioning**. Check the [bundling & minification document](../UI/AspNetCore/Bundling-Minification.md) for details.
* ``abp-card` is a tag helper for Twitter Bootstrap's [card component](https://getbootstrap.com/docs/4.5/components/card/). There are other useful tag helpers provided by the ABP Framework to easily use most of [bootstrap](https://getbootstrap.com/)s components. You could use the regular HTML tags instead of these tag helpers, but using tag helpers reduces HTML code and prevents errors by the help of the IntelliSense and compiles time type checking. For further information, check the [tag helpers](../UI/AspNetCore/Tag-Helpers/Index.md) document.
```

Index.js

Create an `Index.js` file under the `Pages/Books` folder:

![bookstore-index-js-file](images/bookstore-index-js-file-v3.png)

The content of the file is shown below:

```

```js
$(function () {
 var l = abp.localization.getResource('BookStore');

 var dataTable = $('#BooksTable').DataTable(
 abp.libs.datatables.normalizeConfiguration({
 serverSide: true,
 paging: true,
 order: [[1, "asc"]],
 searching: false,
 scrollX: true,
 ajax:
 abp.libs.datatables.createAjax(acme.bookStore.books.book.getList),
 columnDefs: [
 {
 title: l('Name'),
 data: "name"
 },
 {
 title: l('Type'),

```

```

 data: "type",
 render: function (data) {
 return l('Enum:BookType.' + data);
 }
 },
{
 title: l('PublishDate'),
 data: "publishDate",
 render: function (data) {
 return luxon
 .DateTime
 .fromISO(data, {
 locale: abp.localization.currentCulture.name
 }).toLocaleString();
 }
},
{
 title: l('Price'),
 data: "price"
},
{
 title: l('CreationTime'), data: "creationTime",
 render: function (data) {
 return luxon
 .DateTime
 .fromISO(data, {
 locale: abp.localization.currentCulture.name
 }).toLocaleString(luxon.DateTime.DATETIME_SHORT);
 }
}
])
);
}
);
});
```

- \* `'abp.localization.getResource'` gets a function that is used to localize text using the same JSON file defined on the server side. In this way, you can share the localization values with the client side.
- \* `'abp.libs.datatables.normalizeConfiguration'` is a helper function defined by the ABP Framework. There's no requirement to use it, but it simplifies the [\[Datatables\]\(https://datatables.net/\)](https://datatables.net/) configuration by providing conventional default values for missing options.
- \* `'abp.libs.datatables.createAjax'` is another helper function to adapt the ABP's dynamic JavaScript API proxies to the [\[Datatable\]\(https://datatables.net/\)](https://datatables.net/)'s expected parameter format
- \* `'acme.bookStore.books.book.getList'` is the dynamic JavaScript proxy function introduced before.
- \* `[luxon](https://moment.github.io/luxon/)` library is also a standard library that is pre-configured in the solution, so you can use to perform date/time operations easily.

> See [Datatables documentation](<https://datatables.net/manual/>) for all configuration options.

## ## Run the Final Application

You can run the application! The final UI of this part is shown below:

![Book list](images/bookstore-book-list-4.png)

This is a fully working, server side paged, sorted and localized table of books.

```
{{else if UI == "NG"}}
```

```
Install NPM packages
```

> Notice: This tutorial is based on the ABP Framework v3.1.0+. If your project version is older, then please upgrade your solution. Check the [migration guide](../UI/Angular/Migration-Guide-v3.md) if you are upgrading an existing project with v2.x.

If you haven't done it before, open a new command line interface (terminal window) and go to your 'angular' folder and then run the 'yarn' command to install the NPM packages:

```
```bash
yarn
````
```

```
Create a Books Page
```

It's time to create something visible and usable! There are some tools that we will use when developing the Angular frontend application:

- [Ng Bootstrap](https://ng-bootstrap.github.io/#/home) will be used as the UI component library.
- [Ngx-Datatable](https://swimlane.gitbook.io/ngx-datatable/) will be used as the datatable library.

Run the following command line to create a new module, named 'BookModule' in the root folder of the angular application:

```
```bash
yarn ng generate module book --module app --routing --route books
````
```

This command should produce the following output:

```
```bash
> yarn ng generate module book --module app --routing --route books

yarn run v1.19.1
$ ng generate module book --module app --routing --route books
CREATE src/app/book/book-routing.module.ts (336 bytes)
CREATE src/app/book/book.module.ts (335 bytes)
CREATE src/app/book/book.component.html (19 bytes)
CREATE src/app/book/book.component.spec.ts (614 bytes)
CREATE src/app/book/book.component.ts (268 bytes)
CREATE src/app/book/book.component.scss (0 bytes)
UPDATE src/app/app-routing.module.ts (1289 bytes)
Done in 3.88s.
````
```

```
BookModule
```

Open the `/src/app/book/book.module.ts` and replace the content as shown below:

```
```js
import { NgModule } from '@angular/core';
import { SharedModule } from '../shared/shared.module';
import { BookRoutingModule } from './book-routing.module';
import { BookComponent } from './book.component';

@NgModule({
  declarations: [BookComponent],
  imports: [
    BookRoutingModule,
    SharedModule
  ]
})
export class BookModule { }

```
* Added the `SharedModule`. `SharedModule` exports some common modules needed to create user interfaces.
* `SharedModule` already exports the `CommonModule`, so we've removed the `CommonModule`.

Routing
```

The generated code places the new route definition to the `src/app/app-routing.module.ts` file as shown below:

```
```js
const routes: Routes = [
  // other route definitions...
  { path: 'books', loadChildren: () => import('./book/book.module').then(m =>
m.BookModule) },
];
```

```

Now, open the `src/app/route.provider.ts` file and replace the `configureRoutes` function declaration as shown below:

```
```js
function configureRoutes(routes: RoutesService) {
  return () => {
    routes.add([
      {
        path: '/',
        name: '::Menu:Home',
        iconClass: 'fas fa-home',
        order: 1,
        layout: eLayoutType.application,
      },
      {
        path: '/book-store',
        name: '::Menu:BookStore',
        iconClass: 'fas fa-book',
        order: 2,
      }
    ]);
  };
}
```

```

```

 layout: eLayoutType.application,
 },
 {
 path: '/books',
 name: ':::Menu:Books',
 parentName: ':::Menu:BookStore',
 layout: eLayoutType.application,
 },
]);
};

}...

```

`'RoutesService'` is a service provided by the ABP Framework to configure the main menu and the routes.

- \* `'path'` is the URL of the route.
- \* `'name'` is the localized menu item name (check the [[localization document](#)](../UI/Angular/Localization.md) for details).
- \* `'iconClass'` is the icon of the menu item (you can use [[Font Awesome](#)](https://fontawesome.com/) icons by default).
- \* `'order'` is the order of the menu item.
- \* `'layout'` is the layout of the BooksModule's routes (there are three types of pre-defined layouts: `'eLayoutType.application'`, `'eLayoutType.account'` or `'eLayoutType.empty'`).

For more information, check the [[RoutesService document](#)](../UI/Angular/Modifying-the-Menu.md#via-routesservice).

### ### Service Proxy Generation

[[ABP CLI](#)](../CLI.md) provides a `'generate-proxy'` command that generates client proxies for your HTTP APIs to make your HTTP APIs easy to consume by the client side. Before running the `'generate-proxy'` command, your host must be up and running.

> \*\*Warning\*\*: There is a problem with IIS Express; it doesn't allow connecting to the application from another process. If you are using Visual Studio, select the `'Acme.BookStore.HttpApi.Host'` instead of IIS Express in the run button drop-down list, as shown in the figure below:

![[vs-run-without-iisexpress](#)](images/vs-run-without-iisexpress.png)

Once the host application is running, execute the following command in the `'angular'` folder:

```
```bash
abp generate-proxy -t ng
```

```

This command will create the following files under the `'/src/app/proxy/books'` folder:

![[Generated files](#)](images/generated-proxies-3.png)

### ### BookComponent

Open the `/src/app/book/book.component.ts` file and replace the content as below:

```
```js
import { ListService, PagedResultDto } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';
import { BookService, BookDto } from '@proxy/books';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.scss'],
  providers: [ListService],
})
export class BookComponent implements OnInit {
  book = { items: [], totalCount: 0 } as PagedResultDto<BookDto>;

  constructor(public readonly list: ListService, private bookService: BookService) {}

  ngOnInit() {
    const bookStreamCreator = (query) => this.bookService.getList(query);

    this.list.hookToQuery(bookStreamCreator).subscribe((response) => {
      this.book = response;
    });
  }
}

* We imported and injected the generated `BookService`.
* We are using the [ListService](../UI/Angular/List-Service.md), a utility service from the ABP Framework which provides easy pagination, sorting and searching.
```

Open the `/src/app/book/book.component.html` and replace the content as shown below:

```
```html
<div class="card">
 <div class="card-header">
 <div class="row">
 <div class="col col-md-6">
 <h5 class="card-title">
 {{::Menu:Books | abpLocalization}}
 </h5>
 </div>
 <div class="text-end col col-md-6"></div>
 </div>
 </div>
 <div class="card-body">
 <ngx-datatable [rows]="book.items" [count]="book.totalCount"
 [list]="list" default>
 <ngx-datatable-column [name]="'::Name' | abpLocalization"
 prop="name"></ngx-datatable-column>
 <ngx-datatable-column [name]="'::Type' | abpLocalization" prop="type">
 <ng-template let-row="row" ngx-datatable-cell-template>
 {{::Enum:BookType.' + row.type | abpLocalization}}
 </ng-template>
 </ngx-datatable-column>
 </ngx-datatable>
 </div>
</div>
```

```

 </ng-template>
 </ngx-datatable-column>
 <ngx-datatable-column [name]="'::PublishDate' | abpLocalization"
prop="publishDate">
 <ng-template let-row="row" ngx-datatable-cell-template>
 {{row.publishDate | date}}
 </ng-template>
 </ngx-datatable-column>
 <ngx-datatable-column [name]="'::Price' | abpLocalization"
prop="price">
 <ng-template let-row="row" ngx-datatable-cell-template>
 {{row.price | currency}}
 </ng-template>
 </ngx-datatable-column>
</ngx-datatable>
</div>
</div>
```

```

Now you can see the final result on your browser:

```
![Book list final result](images/bookstore-book-list-angular.png)

{{else if UI == "Blazor" || UI == "BlazorServer"}}

## Create a Books Page
```

It's time to create something visible and usable! Right click on the `Pages` folder under the `Acme.BookStore.Blazor` project and add a new **razor component**, named `Books.razor`:

```
![blazor-add-books-component](images/blazor-add-books-component.png)
```

Replace the contents of this component as shown below:

```
```html
@page "/books"

<h2>Books</h2>

@code {
}

```

```

Add the Books Page to the Main Menu

Open the `BookStoreMenuContributor` class in the `Blazor` project add the following code to the end of the `ConfigureMainMenuAsync` method:

```
```csharp
context.Menu.AddItem(
 new ApplicationMenuItem(
 "BooksStore",
 l["Menu:BookStore"],
 icon: "fa fa-book"
).AddItem(
 new ApplicationMenuItem(

```

```

 "BooksStore.Books",
 l["Menu:Books"],
 url: "/books"
)
)
);

```

Run the project, login to the application with the username `admin` and the password `1q2w3E\*` and see that the new menu item has been added to the main menu:

![blazor-menu-bookstore](images/bookstore-new-menu-item-2.png)

When you click on the Books menu item under the Book Store parent, you will be redirected to the new empty Books Page.

### ### Book List

We will use the [Blazorise library](https://blazorise.com/) as the UI component kit. It is a very powerful library that supports major HTML/CSS frameworks, including Bootstrap.

ABP Framework provides a generic base class - `AbpCrudPageBase<...>`, to create CRUD style pages. This base class is compatible with the `ICrudAppService` that was used to build the `IBookAppService`. So, we can inherit from the `AbpCrudPageBase` to automate the code behind for the standard CRUD stuff.

Open the `Books.razor` and replace the content as the following:

```

```xml
@page "/books"
@using Volo.Abp.Application.Dtos
@using Acme.BookStore.Books
@using Acme.BookStore.Localization
@using Microsoft.Extensions.Localization
@inject IStringLocalizer<BookStoreResource> L
@inherits AbpCrudPageBase<IBookAppService, BookDto, Guid,
PagedAndSortedResultRequestDto, CreateUpdateBookDto>

<Card>
    <CardHeader>
        <h2>@L["Books"]</h2>
    </CardHeader>
    <CardBody>
        <DataGrid TItem="BookDto"
            Data="Entities"
            ReadData="OnDataGridReadAsync"
            TotalItems="TotalCount"
            ShowPager="true"
            PageSize="PageSize">
            <DataGridColumn TItem="BookDto"
                Field="@nameof(BookDto.Name)"
                Caption="@L["Name"]"></DataGridColumn>
            <DataGridColumn TItem="BookDto"
                Field="@nameof(BookDto.Type)">

```

```

                Caption="@L["Type"]">
            <DisplayTemplate>
                @L[$"Enum:BookType.{context.Type}"]
            </DisplayTemplate>
        </DataGridColumn>
        <DataGridColumn TItem="BookDto"
                        Field="@nameof(BookDto.PublishDate)"
                        Caption="@L["PublishDate"]">
            <DisplayTemplate>
                @context.PublishDate.ToShortDateString()
            </DisplayTemplate>
        </DataGridColumn>
        <DataGridColumn TItem="BookDto"
                        Field="@nameof(BookDto.Price)"
                        Caption="@L["Price"]">
        </DataGridColumn>
        <DataGridColumn TItem="BookDto"
                        Field="@nameof(BookDto.CreationTime)"
                        Caption="@L["CreationTime"]">
            <DisplayTemplate>
                @context.CreationTime.ToString("yyyy-MM-dd")
            </DisplayTemplate>
        </DataGridColumn>
    </DataGridColumns>
</DataGrid>
</CardBody>
</Card>
```

```

> If you see some syntax errors, you can ignore them if your application is properly built and running. Visual Studio still has some bugs with Blazor.

- \* Inherited from `AbpCrudPageBase<IBookAppService, BookDto, Guid, PagedAndSortedResultRequestDto, CreateUpdateBookDto>` which implements all the CRUD details for us.
- \* `Entities`, `TotalCount`, `PageSize`, `OnDataGridReadAsync` are defined in the base class.
- \* Injected `IStringLocalizer<BookStoreResource>` (as `L` object) and used for localization.

While the code above is pretty easy to understand, you can check the Blazorise [Card](<https://blazorise.com/docs/components/card/>) and [DataGrid](<https://blazorise.com/docs/extensions/datagrid/>) documents to understand them better.

#### #### About the AbpCrudPageBase

We will continue benefitting from `AbpCrudPageBase` for the books page. You could just inject the `IBookAppService` and perform all the server side calls yourself (thanks to the [Dynamic C# HTTP API Client Proxy]([./API/Dynamic-CSharp-API-Clients.md](#)) system of the ABP Framework). We will do it manually for the authors page to demonstrate how to call the server side HTTP APIs in your Blazor applications.

#### ## Run the Final Application

You can run the application! The final UI of this part is shown below:

![blazor-bookstore-book-list](images/blazor-bookstore-book-list-2.png)

This is a fully working, server side paged, sorted and localized table of books.

{}{{end # UI }}

## The Next Part

Check the [next part](Part-3.md) of this tutorial.

### 3.1.3 3: Creating, Updating and Deleting Books

```
Web Application Development Tutorial - Part 3: Creating, Updating and
Deleting Books
```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```
About This Tutorial
```

In this tutorial series, you will build an ABP based web application named `Acme.BookStore`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* \*\*{{DB\_Value}}\*\* as the ORM provider.
- \* \*\*{{UI\_Value}}\*\* as the UI Framework.

This tutorial is organized as the following parts:

- [Part 1: Creating the server side](Part-1.md)
- [Part 2: The book list page](Part-2.md)
- \*\*Part 3: Creating, updating and deleting books (this part)\*\*
- [Part 4: Integration tests](Part-4.md)
- [Part 5: Authorization](Part-5.md)
- [Part 6: Authors: Domain layer](Part-6.md)
- [Part 7: Authors: Database Integration](Part-7.md)
- [Part 8: Authors: Application Layer](Part-8.md)
- [Part 9: Authors: User Interface](Part-9.md)
- [Part 10: Book to Author Relation](Part-10.md)

### Download the Source Code

This tutorial has multiple versions based on your \*\*UI\*\* and \*\*Database\*\* preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [MVC (Razor Pages) UI with EF Core](https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore)
- \* [Blazor UI with EF Core](https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore)
- \* [Angular UI with MongoDB](https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb)

```
> If you encounter the "filename too long" or "unzip" error on Windows,
please see [this guide](../KB/Windows-Path-Too-Long-Fix.md).

{{if UI == "MVC" && DB == "EF"}}

Video Tutorial
```

This part is also recorded as a video tutorial and \*\*<a href="https://www.youtube.com/watch?v=TLShZ08u2VE&list=PLsNclT2aHJcPNaCf7Io3D bMN6yAk\_DgWJ&index=3" target="\_blank">published on YouTube</a>\*\*.

```
 {{end}}

{{if UI == "MVC"}}

Creating a New Book
```

In this section, you will learn how to create a new modal dialog form to create a new book. The modal dialog will look like the image below:

```
![bookstore-create-dialog](./images/bookstore-create-dialog-3.png)
```

```
Create the Modal Form
```

Create a new razor page named `CreateModal.cshtml` under the `Pages/Books` folder of the `Acme.BookStore.Web` project.

```
![bookstore-add-create-dialog](./images/bookstore-add-create-dialog-v2.png)
```

```
CreateModal.cshtml.cs
```

Open the `CreateModal.cshtml.cs` file (`CreateModalModel` class) and replace it with the following code:

```
```C#  
using System.Threading.Tasks;  
using Acme.BookStore.Books;  
using Microsoft.AspNetCore.Mvc;  
  
namespace Acme.BookStore.Web.Pages.Books  
{  
    public class CreateModalModel : BookStorePageModel  
    {  
        [BindProperty]  
        public CreateUpdateBookDto Book { get; set; }  
  
        private readonly IBookAppService _bookAppService;  
  
        public CreateModalModel(IBookAppService bookAppService)  
        {  
            _bookAppService = bookAppService;  
        }  
  
        public void OnGet()  
        {  
            Book = new CreateUpdateBookDto();  
        }  
}
```

```

        public async Task<IActionResult> OnPostAsync()
    {
        await _bookAppService.CreateAsync(Book);
        return NoContent();
    }
}

...

```

* This class is derived from the `BookStorePageModel` instead of the standard `PageModel`. `BookStorePageModel` indirectly inherits the `PageModel` and adds some common properties & methods that can be shared in your page model classes.

- * `[BindProperty]` attribute on the `Book` property binds post request data to this property.
- * This class simply injects the `IBookAppService` in the constructor and calls the `CreateAsync` method in the `OnPostAsync` handler.
- * It creates a new `CreateUpdateBookDto` object in the `OnGet` method. ASP.NET Core can work without creating a new instance like that. However, it doesn't create an instance for you and if your class has some default value assignments or code execution in the class constructor, they won't work. For this case, we set default values for some of the `CreateUpdateBookDto` properties.

CreateModal.cshtml

Open the `CreateModal.cshtml` file and paste the code below:

```

```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Modal
@model CreateModalModel
@inject IStringLocalizer<BookStoreResource> L
 @{
 Layout = null;
}
<abp-dynamic-form abp-model="Book" asp-page="/Books/CreateModal">
 <abp-modal>
 <abp-modal-header title="@L["NewBook"].Value"></abp-modal-header>
 <abp-modal-body>
 <abp-form-content />
 </abp-modal-body>
 <abp-modal-footer
buttons="@((AbpModalButtons.Cancel | AbpModalButtons.Save))"></abp-modal-footer>
 </abp-modal>
</abp-dynamic-form>
```

```

* This modal uses `abp-dynamic-form` [tag helper](./UI/AspNetCore/Tag-Helpers/Dynamic-Forms.md) to automatically create the form from the `CreateUpdateBookDto` model class.

* `abp-model` attribute indicates the model object where it's the `Book` property in this case.

* `abp-form-content` tag helper is a placeholder to render the form controls (it is optional and needed only if you have added some other content in the `abp-dynamic-form` tag, just like in this page).

> Tip: `Layout` should be `null` just as done in this example since we don't want to include all the layout for the modals when they are loaded via AJAX.

Add the "New book" Button

Open the `Pages/Books/Index.cshtml` and set the content of `abp-card-header` tag as below:

```
```html
<abp-card-header>
 <abp-row>
 <abp-column size-md="_6">
 <abp-card-title>@L["Books"]</abp-card-title>
 </abp-column>
 <abp-column size-md="_6" class="text-end">
 <abp-button id="NewBookButton"
 text="@L["NewBook"].Value"
 icon="plus"
 button-type="Primary"/>
 </abp-column>
 </abp-row>
</abp-card-header>
```
```

The final content of `Index.cshtml` is shown below:

```
```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@model IndexModel
@inject IStringLocalizer<BookStoreResource> L
@section scripts
{
 <abp-script src="/Pages/Books/Index.js"/>
}

<abp-card>
 <abp-card-header>
 <abp-row>
 <abp-column size-md="_6">
 <abp-card-title>@L["Books"]</abp-card-title>
 </abp-column>
 <abp-column size-md="_6" class="text-end">
 <abp-button id="NewBookButton"
 text="@L["NewBook"].Value"
 icon="plus"
 button-type="Primary"/>
 </abp-column>
 </abp-row>
 </abp-card-header>
 <abp-card-body>
 <abp-table striped-rows="true" id="BooksTable"></abp-table>
 </abp-card-body>

```

```
</abp-card-body>
</abp-card>
```
```

This adds a new button called ****New book**** to the ****top-right**** of the table:

```
![bookstore-new-book-button](./images/bookstore-new-book-button-3.png)
```

Open the `'Pages/Books/Index.js'` file and add the following code right after the `'Datatable'` configuration:

```
```js
var createModal = new abp.ModalManager(abp.appPath + 'Books/CreateModal');

createModal.onResult(function () {
 dataTable.ajax.reload();
});

$('#NewBookButton').click(function (e) {
 e.preventDefault();
 createModal.open();
});
```

* 'abp.ModalManager' is a helper class to manage modals on the client side. It internally uses Twitter Bootstrap's standard modal, but abstracts many details by providing a simple API.
* 'createModal.onResult(...)' used to refresh the data table after creating a new book.
* 'createModal.open();' is used to open the model to create a new book.
```

The final content of the `'Index.js'` file should be like this:

```
```js
$(function () {
 var l = abp.localization.getResource('BookStore');

 var dataTable = $('#BooksTable').DataTable(
 abp.libs.datatables.normalizeConfiguration({
 serverSide: true,
 paging: true,
 order: [[1, "asc"]],
 searching: false,
 scrollX: true,
 ajax:
 abp.libs.datatables.createAjax(acme.bookStore.books.book.getList),
 columnDefs: [
 {
 title: l('Name'),
 data: "name"
 },
 {
 title: l('Type'),
 data: "type",
 render: function (data) {
 return l('Enum:BookType.' + data);
 }
 }
]
);
});
```

```

 {
 title: l('PublishDate'),
 data: "publishDate",
 dataFormat: "datetime"
 },
 {
 title: l('Price'),
 data: "price"
 },
 {
 title: l('CreationTime'), data: "creationTime",
 dataFormat: "datetime"
 }
]
)
);

var createModal = new abp.ModalManager(abp.appPath +
'Books/CreateModal');

createModal.onResult(function () {
 dataTable.ajax.reload();
});

$('#NewBookButton').click(function (e) {
 e.preventDefault();
 createModal.open();
});
});
}

```

Now, you can \*\*run the application\*\* and add some new books using the new modal form.

#### ## Updating a Book

Create a new razor page, named `EditModal.cshtml` under the `Pages/Books` folder of the `Acme.BookStore.Web` project:

![bookstore-add-edit-dialog](./images/bookstore-add-edit-dialog.png)

#### ### EditModal.cshtml.cs

Open the `EditModal.cshtml.cs` file (`EditModalModel` class) and replace it with the following code:

```

```csharp
using System;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;

namespace Acme.BookStore.Web.Pages.Books;

public class EditModalModel : BookStorePageModel
{
    [HiddenInput]
    [BindProperty(SupportsGet = true)]

```

```

public Guid Id { get; set; }

[BindProperty]
public CreateUpdateBookDto Book { get; set; }

private readonly IBookAppService _bookAppService;

public EditModalModel(IBookAppService bookAppService)
{
    _bookAppService = bookAppService;
}

public async Task OnGetAsync()
{
    var bookDto = await _bookAppService.GetAsync(Id);
    Book = ObjectMapper.Map<BookDto, CreateUpdateBookDto>(bookDto);
}

public async Task<IActionResult> OnPostAsync()
{
    await _bookAppService.UpdateAsync(Id, Book);
    return NoContent();
}
}

```


- `'[HiddenInput]'` and `'[BindProperty]'` are standard ASP.NET Core MVC attributes. ``SupportsGet` is used to be able to get the ``Id` value from the query string parameter of the request.
- In the ``OnGetAsync` method, we get the ``BookDto` from the ``BookAppService` and this is being mapped to the DTO object ``CreateUpdateBookDto`.
- The ``OnPostAsync` uses ``BookAppService.UpdateAsync(...)` to update the entity.

```

#### ### Mapping from BookDto to CreateUpdateBookDto

To be able to map the ``BookDto` to ``CreateUpdateBookDto` , configure a new mapping. To do this, open the ``BookStoreWebAutoMapperProfile.cs` file in the ``Acme.BookStore.Web` project and change it as shown below:

```

```csharp
using AutoMapper;

namespace Acme.BookStore.Web;

public class BookStoreWebAutoMapperProfile : Profile
{
    public BookStoreWebAutoMapperProfile()
    {
        CreateMap<BookDto, CreateUpdateBookDto>();
    }
}
```

```

\* We have just added ``CreateMap<BookDto, CreateUpdateBookDto>();` to define this mapping.

> Notice that we do the mapping definition in the web layer as a best practice since it is only needed in this layer.

### ### EditModal.cshtml

Replace `EditModal.cshtml` content with the following content:

```
```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Modal
@model EditModalModel
@inject IStringLocalizer<BookStoreResource> L
 @{
     Layout = null;
}
<abp-dynamic-form abp-model="Book" asp-page="/Books/EditModal">
    <abp-modal>
        <abp-modal-header title="@L["Update"].Value"></abp-modal-header>
        <abp-modal-body>
            <abp-input asp-for="Id" />
            <abp-form-content />
        </abp-modal-body>
        <abp-modal-footer
buttons="@((AbpModalButtons.Cancel | AbpModalButtons.Save))"></abp-modal-footer>
    </abp-modal>
</abp-dynamic-form>
````
```

This page is very similar to `CreateModal.cshtml`, except:

- \* It includes an `abp-input` for the `Id` property to store the `Id` of the editing book (which is a hidden input).
- \* It uses `Books/EditModal` as the post URL.

### ### Add "Actions" Dropdown to the Table

We will add a dropdown button to the table named *\*Actions\**.

Open the `Pages/Books/Index.js` file and replace the content as below:

```
```js
$(function () {
    var l = abp.localization.getResource('BookStore');
    var createModal = new abp.ModalManager(abp.appPath +
'Books/CreateModal');
    var editModal = new abp.ModalManager(abp.appPath + 'Books/EditModal');

    var dataTable = $('#BooksTable').DataTable(
        abp.libs.datatables.normalizeConfiguration({
            serverSide: true,
            paging: true,
            order: [[1, "asc"]],
            searching: false,
            scrollX: true,
```

```

        ajax:
    abp.libs.datatables.createAjax(acme.bookStore.books.book.getList),
    columnDefs: [
        {
            title: l('Actions'),
            rowAction: {
                items:
                    [
                        {
                            text: l('Edit'),
                            action: function (data) {
                                editModal.open({ id:
data.record.id });
                            }
                        }
                    ]
            }
        },
        {
            title: l('Name'),
            data: "name"
        },
        {
            title: l('Type'),
            data: "type",
            render: function (data) {
                return l('Enum:BookType.' + data);
            }
        },
        {
            title: l('PublishDate'),
            data: "publishDate",
            dataFormat: "datetime"
        },
        {
            title: l('Price'),
            data: "price"
        },
        {
            title: l('CreationTime'), data: "creationTime",
            dataFormat: "datetime"
        }
    ]
})
);

createModal.onResult(function () {
    dataTable.ajax.reload();
});

editModal.onResult(function () {
    dataTable.ajax.reload();
});

$('#NewBookButton').click(function (e) {
    e.preventDefault();
    createModal.open();
});

```

```

});  
  

* Added a new `ModalManager` named `editModal` to open the edit modal dialog.  

* Added a new column at the beginning of the `columnDefs` section. This  

  column is used for the "*Actions*" dropdown button.  

* The "*Edit*" action simply calls `editModal.open()` to open the edit  

  dialog.  

* The `editModal.onResult(...)` callback refreshes the data table when you  

  close the edit modal.

```

You can run the application and edit any book by selecting the edit action on a book.

The final UI looks as below:

![bookstore-books-table-actions](./images/bookstore-edit-button-3.png)

> Notice that you don't see the "Actions" button in the figure below. Instead, you see an "Edit" button. ABP is smart enough to show a single simple button instead of a actions dropdown button when the dropdown has only a single item. After the next section, it will turn to a drop down button.

Deleting a Book

Open the `Pages/Books/Index.js` file and add a new item to the `rowAction` `items`:

```

```js
{
 text: l('Delete'),
 confirmMessage: function (data) {
 return l('BookDeletionConfirmationMessage', data.record.name);
 },
 action: function (data) {
 acme.bookStore.books.book
 .delete(data.record.id)
 .then(function() {
 abp.notify.info(l('SuccessfullyDeleted'));
 dataTable.ajax.reload();
 });
 }
}...
```

```

* The `confirmMessage` option is used to ask a confirmation question before executing the `action`.
* The `acme.bookStore.books.book.delete(...)` method makes an AJAX request to the server to delete a book.
* `abp.notify.info()` shows a notification after the delete operation.

Since we've used two new localization texts (`BookDeletionConfirmationMessage` and `SuccessfullyDeleted`) you need to add these to the localization file (`en.json` under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project):

```

```json
"BookDeletionConfirmationMessage": "Are you sure to delete the book '{0}'?",
```

"SuccessfullyDeleted": "Successfully deleted!"

The final 'Index.js' content is shown below:

```

 title: l('Name'),
 data: "name"
 },
 {
 title: l('Type'),
 data: "type",
 render: function (data) {
 return l('Enum:BookType.' + data);
 }
 },
 {
 title: l('PublishDate'),
 data: "publishDate",
 dataFormat: "datetime"
 },
 {
 title: l('Price'),
 data: "price"
 },
 {
 title: l('CreationTime'), data: "creationTime",
 dataFormat: "datetime"
 }
]
})
);

createModal.onResult(function () {
 dataTable.ajax.reload();
});

editModal.onResult(function () {
 dataTable.ajax.reload();
});

$('#NewBookButton').click(function (e) {
 e.preventDefault();
 createModal.open();
});
});
}
);

```

You can run the application and try to delete a book.

```

{{end}}

{{if UI == "NG"}}

Creating a New Book

```

In this section, you will learn how to create a new modal dialog form to create a new book.

### ### BookComponent

Open `/src/app/book/book.component.ts` and replace the content as below:

```
'''js
```

```

import { ListService, PagedResultDto } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';
import { BookService, BookDto } from '@proxy/books';

@Component({
 selector: 'app-book',
 templateUrl: './book.component.html',
 styleUrls: ['./book.component.scss'],
 providers: [ListService],
})
export class BookComponent implements OnInit {
 book = { items: [], totalCount: 0 } as PagedResultDto<BookDto>;
 isModalOpen = false; // add this line

 constructor(public readonly list: ListService, private bookService: BookService) {}

 ngOnInit() {
 const bookStreamCreator = (query) => this.bookService.getList(query);

 this.list.hookToQuery(bookStreamCreator).subscribe((response) => {
 this.book = response;
 });
 }

 // add new method
 createBook() {
 this.isModalOpen = true;
 }
}

* We defined a property called `isModalOpen` and a method called `createBook`.

```

Open `/src/app/book/book.component.html` and make the following changes:

```

```html


<div class="card-header">
    <div class="row">
      <div class="col col-md-6">
        <h5 class="card-title">{${{ 'Menu:Books' | abpLocalization }}}</h5>
      </div>
      <div class="text-end col col-md-6">

        <!-- Add the "new book" button here -->
        <div class="text-lg-end pt-2">
          <button id="create" class="btn btn-primary" type="button"
(click)="createBook()">
            <i class="fa fa-plus me-1"></i>
            <span>{${{ 'NewBook' | abpLocalization }}}</span>
          </button>
        </div>
      </div>
    </div>
  </div>


```

```

        </div>
    </div>
</div>
<div class="card-body">
    <!-- ngx-datatable should be here! -->
</div>
</div>

<!-- Add the modal here -->
<abp-modal [(visible)]="isModalOpen">
    <ng-template #abpHeader>
        <h3>{${{ 'NewBook' | abpLocalization }}}</h3>
    </ng-template>

    <ng-template #abpBody> </ng-template>

    <ng-template #abpFooter>
        <button type="button" class="btn btn-secondary" abpClose>
            ${{ 'Close' | abpLocalization }}
        </button>
    </ng-template>
</abp-modal>
```
* Added a 'New book' button to the card header..
* Added the `abp-modal` which renders a modal to allow user to create a new book. `abp-modal` is a pre-built component to show modals. While you could use another approach to show a modal, `abp-modal` provides additional benefits.
```

You can open your browser and click the \*\*New book\*\* button to see the new modal.

![Empty modal for new book](./images/bookstore-empty-new-book-modal-2.png)

### ## Create a Reactive Form

[Reactive forms](https://angular.io/guide/reactive-forms) provide a model-driven approach to handling form inputs whose values change over time.

Open `/src/app/book/book.component.ts` and replace the content as below:

```

```js
import { ListService, PagedResultDto } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';
import { BookService, BookDto, bookTypeOptions } from '@proxy/books'; // add bookTypeOptions
import { FormGroup, FormBuilder, Validators } from '@angular/forms'; // add this

@Component({
    selector: 'app-book',
    templateUrl: './book.component.html',
    styleUrls: ['./book.component.scss'],
    providers: [ListService],
})
export class BookComponent implements OnInit {
    book = { items: [], totalCount: 0 } as PagedResultDto<BookDto>;
}
```

```

```

form: FormGroup; // add this line

// add bookTypes as a list of BookType enum members
bookTypes = bookTypeOptions;

isModalOpen = false;

constructor(
 public readonly list: ListService,
 private bookService: BookService,
 private fb: FormBuilder // inject FormBuilder
) {}

ngOnInit() {
 const bookStreamCreator = (query) => this.bookService.getList(query);

 this.list.hookToQuery(bookStreamCreator).subscribe((response) => {
 this.book = response;
 });
}

createBook() {
 this.buildForm(); // add this line
 this.isModalOpen = true;
}

// add buildForm method
buildForm() {
 this.form = this.fb.group({
 name: ['', Validators.required],
 type: [null, Validators.required],
 publishDate: [null, Validators.required],
 price: [null, Validators.required],
 });
}

// add save method
save() {
 if (this.form.invalid) {
 return;
 }

 this.bookService.create(this.form.value).subscribe(() => {
 this.isModalOpen = false;
 this.form.reset();
 this.list.get();
 });
}
```
}

* Imported `FormGroup`, `FormBuilder` and `Validators` from `@angular/forms`.
* Added a `form: FormGroup` property.
* Added a `bookTypes` property as a list of `BookType` enum members. That will be used in form options.
* Injected `FormBuilder` into the constructor.
[FormBuilder](https://angular.io/api/forms/FormBuilder) provides convenient
```

methods for generating form controls. It reduces the amount of boilerplate needed to build complex forms.

- * Added a `buildForm` method to the end of the file and executed the `buildForm()` in the `createBook` method.
- * Added a `save` method.

Open `/src/app/book/book.component.html` and replace `</ng-template>` with the following code part:

```
```html
<ng-template #abpBody>
 <form [formGroup]="form" (ngSubmit)="save()">
 <div class="mt-2">
 <label for="book-name">Name</label> *
 <input type="text" id="book-name" class="form-control"
formControlName="name" autofocus />
 </div>

 <div class="mt-2">
 <label for="book-price">Price</label> *
 <input type="number" id="book-price" class="form-control"
formControlName="price" />
 </div>

 <div class="mt-2">
 <label for="book-type">Type</label> *
 <select class="form-control" id="book-type" formControlName="type">
 <option [ngValue]="">Select a book type</option>
 <option [ngValue]="${type.value}" *ngFor="let type of bookTypes">
 ${'::Enum:BookType.' + type.value} | abpLocalization
 </option>
 </select>
 </div>

 <div class="mt-2">
 <label>Publish date</label> *
 <input
 #datepicker="ngbDatepicker"
 class="form-control"
 name="datepicker"
 formControlName="publishDate"
 ngbDatepicker
 (click)="datepicker.toggle()"/>
 </div>
 </form>
</ng-template>
```

```

Also replace `</ng-template>` with the following code part:

```
```html
<ng-template #abpFooter>
 <button type="button" class="btn btn-secondary" abpClose>
 ${'::Close' | abpLocalization}
 </button>

 <!--added save button-->

```

```

<button class="btn btn-primary" (click)="save()" [disabled]="form.invalid">
 <i class="fa fa-check mr-1"></i>
 {${{ 'Save' | abpLocalization }}}
</button>
</ng-template>
```

```

Datepicker

We've used **[NgBootstrap datepicker]**(<https://ng-bootstrap.github.io/#/components/datepicker/overview>) in this component. So, we need to arrange the dependencies related to this component.

Open `'/src/app/book/book.module.ts` and replace the content as below:

```

```js
import { NgModule } from '@angular/core';
import { SharedModule } from '../shared/shared.module';
import { BookRoutingModule } from './book-routing.module';
import { BookComponent } from './book.component';
import { NgbDatepickerModule } from '@ng-bootstrap/ng-bootstrap'; // add this line

@NgModule({
 declarations: [BookComponent],
 imports: [
 BookRoutingModule,
 SharedModule,
 NgbDatepickerModule, // add this line
]
})
export class BookModule { }
```

```

* We imported `NgbDatepickerModule` to be able to use the date picker.

Open `'/src/app/book/book.component.ts` and replace the content as below:

```

```js
import { ListService, PagedResultDto } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';
import { BookService, BookDto, bookTypeOptions } from '@proxy/books';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';

// added this line
import { NgbDateNativeAdapter, NgbDateAdapter } from '@ng-bootstrap/ng-bootstrap';

@Component({
 selector: 'app-book',
 templateUrl: './book.component.html',
 styleUrls: ['./book.component.scss'],
 providers: [
 ListService,
 { provide: NgbDateAdapter, useClass: NgbDateNativeAdapter } // add this line
],
})
```

```

```

export class BookComponent implements OnInit {
  book = { items: [], totalCount: 0 } as PagedResultDto<BookDto>;
  form: FormGroup;
  bookTypes = bookTypeOptions;
  isModalOpen = false;

  constructor(
    public readonly list: ListService,
    private bookService: BookService,
    private fb: FormBuilder
  ) {}

  ngOnInit() {
    const bookStreamCreator = (query) => this.bookService.getList(query);

    this.list.hookToQuery(bookStreamCreator).subscribe((response) => {
      this.book = response;
    });
  }

  createBook() {
    this.buildForm();
    this.isModalOpen = true;
  }

  buildForm() {
    this.form = this.fb.group({
      name: ['', Validators.required],
      type: [null, Validators.required],
      publishDate: [null, Validators.required],
      price: [null, Validators.required],
    });
  }

  save() {
    if (this.form.invalid) {
      return;
    }

    this.bookService.create(this.form.value).subscribe(() => {
      this.isModalOpen = false;
      this.form.reset();
      this.list.get();
    });
  }
}

```
* Imported `NgbDateNativeAdapter` and `NgbDateAdapter`.
* We added a new provider `NgbDateAdapter` that converts the Datepicker value to `Date` type. Check out the [datepicker adapters](https://ng-bootstrap.github.io/#/components/datepicker/overview) for more details.

```

Now, you can open your browser to see the changes:

![Save button to the modal](./images/bookstore-new-book-form-v3.png)

## ## Updating a Book

Open `./src/app/book/book.component.ts` and replace the content as shown below:

```
```js
import { ListService, PagedResultDto } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';
import { BookService, BookDto, bookTypeOptions } from '@proxy/books';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { NgbDateNativeAdapter, NgbDateAdapter } from '@ng-bootstrap/ng-
bootstrap';

@Component({
  selector: 'app-book',
  templateUrl: './book.component.html',
  styleUrls: ['./book.component.scss'],
  providers: [ListService, { provide: NgbDateAdapter, useClass:
NgbDateNativeAdapter }],
})
export class BookComponent implements OnInit {
  book = { items: [], totalCount: 0 } as PagedResultDto<BookDto>;
  selectedBook = {} as BookDto; // declare selectedBook
  form: FormGroup;
  bookTypes = bookTypeOptions;
  isModalOpen = false;

  constructor(
    public readonly list: ListService,
    private bookService: BookService,
    private fb: FormBuilder
  ) {}

  ngOnInit() {
    const bookStreamCreator = (query) => this.bookService.getList(query);

    this.list.hookToQuery(bookStreamCreator).subscribe((response) => {
      this.book = response;
    });
  }

  createBook() {
    this.selectedBook = {} as BookDto; // reset the selected book
    this.buildForm();
    this.isModalOpen = true;
  }

  // Add editBook method
  editBook(id: string) {
    this.bookService.get(id).subscribe((book) => {
      this.selectedBook = book;
      this.buildForm();
    });
  }
}
```

```

        this.isModalOpen = true;
    });
}

buildForm() {
    this.form = this.fb.group({
        name: [this.selectedBook.name || '', Validators.required],
        type: [this.selectedBook.type || null, Validators.required],
        publishDate: [
            this.selectedBook.publishDate ? new
Date(this.selectedBook.publishDate) : null,
            Validators.required,
        ],
        price: [this.selectedBook.price || null, Validators.required],
    });
}

// change the save method
save() {
    if (this.form.invalid) {
        return;
    }

    const request = this.selectedBook.id
        ? this.bookService.update(this.selectedBook.id, this.form.value)
        : this.bookService.create(this.form.value);

    request.subscribe(() => {
        this.isModalOpen = false;
        this.form.reset();
        this.list.get();
    });
}
```
* We declared a variable named `selectedBook` as `BookDto`.
* We added an `editBook` method. This method fetches the book with the given `id` and sets it to `selectedBook` object.
* We replaced the `buildForm` method so that it creates the form with the `selectedBook` data.
* We replaced the `createBook` method so it sets `selectedBook` to an empty object.
* We changed the `save` method to handle both of create and update operations.

```

### ### Add "Actions" Dropdown to the Table

Open `/src/app/book/book.component.html` and add the following `ngx-datatype-column` definition as the first column in the `ngx-datatype`:

```

```html
<ngx-datatype-column
    [name]="'::Actions' | abpLocalization"
    [maxWidth]="150"
    [sortable]="false"
>
    <ng-template let-row="row" ngx-datatype-cell-template>

```

```

<div ngbDropdown container="body" class="d-inline-block">
  <button
    class="btn btn-primary btn-sm dropdown-toggle"
    data-toggle="dropdown"
    aria-haspopup="true"
    ngbDropdownToggle
  >
    <i class="fa fa-cog me-1"></i>{%%{{'::Actions' | abpLocalization }}}%
  </button>
  <div ngbDropdownMenu>
    <button ngbDropdownItem (click)="editBook(row.id)">
      {%%{{'::Edit' | abpLocalization }}}%
    </button>
  </div>
</div>
</ng-template>
</ngx-datatable-column>
```

```

Added an "Actions" dropdown as the first column of the table that is shown below:

![Action buttons](./images/bookstore-actions-buttons-2.png)

Also, change the `ng-template #abpHeader` section as shown below:

```

```html
<ng-template #abpHeader>
  <h3>{%%{{(selectedBook.id ? '::Edit' : '::NewBook') | abpLocalization }}}%</h3>
</ng-template>
```

```

This template will show the \*\*Edit\*\* text for edit record operation, \*\*New Book\*\* for new record operation in the title.

## ## Deleting a Book

Open the `/src/app/book/book.component.ts` file and inject the `ConfirmationService` .

Replace the constructor as below:

```

```js
// ...

// add new imports
import { ConfirmationService, Confirmation } from '@abp/ng.theme.shared';

//change the constructor
constructor(
  public readonly list: ListService,
  private bookService: BookService,
  private fb: FormBuilder,
  private confirmation: ConfirmationService // inject the ConfirmationService
) {}

```

```

// Add a delete method
delete(id: string) {
  this.confirmation.warn('::AreYouSureToDelete',
    '::AreYouSure').subscribe((status) => {
      if (status === Confirmation.Status.confirm) {
        this.bookService.delete(id).subscribe(() => this.list.get());
      }
    });
}

* We imported `ConfirmationService`.
* We injected `ConfirmationService` to the constructor.
* Added a `delete` method.

> Check out the [Confirmation Popup documentation](./UI/Angular/Confirmation-Service) for more about this service.

```

Add a Delete Button

Open `/src/app/book/book.component.html` and modify the `ngbDropdownMenu` to add the delete button as shown below:

```

```html
<div ngbDropdownMenu>
 <!-- add the Delete button -->
 <button ngbDropdownItem (click)="delete(row.id)">
 {{'Delete' | abpLocalization}}
 </button>
</div>
```

```

The final actions dropdown UI looks like below:

![bookstore-final-actions-dropdown](./images/bookstore-final-actions-dropdown-2.png)

Clicking the "Delete" action calls the `delete` method which then shows a confirmation popup as shown below:

```

![bookstore-confirmation-popup](./images/bookstore-confirmation-popup-2.png)

{{end}}

{{if UI == "Blazor" || UI == "BlazorServer"}}

## Creating a New Book

```

In this section, you will learn how to create a new modal dialog form to create a new book. Since we've inherited from the `AbpCrudPageBase`, we only need to develop the view part.

Add a "New" Button

Open the `Books.razor` and replace the `` section with the following code:

```

```xml
<CardHeader>
 <Row Class="justify-content-between">
 <Column ColumnSize="ColumnSize.IsAuto">
 <h2>@L["Books"]</h2>
 </Column>
 <Column ColumnSize="ColumnSize.IsAuto">
 <Button Color="Color.Primary"
 Clicked="OpenCreateModalAsync">@L["NewBook"]</Button>
 </Column>
 </Row>
</CardHeader>
```

```

This will change the card header by adding a "New book" button to the right side:

![blazor-add-book-button](./images/blazor-add-book-button-2.png)

Now, we can add a modal that will be opened when we click the button.

Book Creation Modal

Open the `Books.razor` and add the following code to the end of the page:

```

```xml
<Modal @ref="@CreateModal">
 <ModalBackdrop />
 <ModalContent IsCentered="true">
 <Form>
 <ModalHeader>
 <ModalTitle>@L["NewBook"]</ModalTitle>
 <CloseButton Clicked="CloseCreateModalAsync"/>
 </ModalHeader>
 <ModalBody>
 <Validations @ref="@CreateValidationsRef" Model="@NewEntity"
ValidateOnLoad="false">
 <Validation MessageLocalizer="@LH.Localize">
 <Field>
 <FieldLabel>@L["Name"]</FieldLabel>
 <TextEdit @bind-Text="@NewEntity.Name">
 <Feedback>
 <ValidationError/>
 </Feedback>
 </TextEdit>
 </Field>
 </Validation>
 <Field>
 <FieldLabel>@L["Type"]</FieldLabel>
 <Select TValue="BookType" @bind-
SelectedValue="@NewEntity.Type">
 @foreach (int bookTypeValue in
Enum.GetValues(typeof(BookType)))
 {
 <SelectItem TValue="BookType"
Value="@((BookType) bookTypeValue)">
 @L[$"Enum:BookType.{bookTypeValue}"]
 </SelectItem>
 }
 </Select>
 </Field>
 </Validations>
 </ModalBody>
 </Form>
 </ModalContent>
</Modal>
```

```

```

                </SelectItem>
            }
        </Select>
    </Field>
    <Field>
        <FieldLabel>@L["PublishDate"]</FieldLabel>
        <DateEdit TValue="DateTime" @bind-
Date="NewEntity.PublishDate"/>
    </Field>
    <Field>
        <FieldLabel>@L["Price"]</FieldLabel>
        <NumericEdit TValue="float" @bind-
Value="NewEntity.Price"/>
    </Field>
    </Validations>
</ModalBody>
<ModalFooter>
    <Button Color="Color.Secondary"
        Clicked="CloseCreateModalAsync">@L["Cancel"]</Button>
    <Button Color="Color.Primary"
        Type="@ButtonType.Submit"
        PreventDefaultOnSubmit="true"
        Clicked="CreateEntityAsync">@L["Save"]</Button>
</ModalFooter>
</Form>
</ModalContent>
</Modal>
```

```

This code requires a service; Inject the `AbpBlazorMessageLocalizerHelper<T>` at the top of the file, just before the `@inherits...` line:

```

```csharp
@inject AbpBlazorMessageLocalizerHelper<BookStoreResource> LH
```

* The form implements validation and the `AbpBlazorMessageLocalizerHelper` is used to simply localize the validation messages.
* The `CreateModal` object, `CloseCreateModalAsync` and `CreateEntityAsync` methods are defined by the base class. Check out the [Blazorise documentation](https://blazorise.com/docs/) if you want to understand the `Modal` and the other components.

```

That's all. Run the application and try to add a new book:

```
![blazor-new-book-modal](./images/blazor-new-book-modal-2.png)
```

### ## Updating a Book

Editing a book is similar to creating a new book.

#### ### Actions Dropdown

Open the `Books.razor` and add the following `DataGridEntityActionsColumn` section inside the `DataGridColumn` as the first item:

```

```xml
<DataGridEntityActionsColumn TItem="BookDto" @ref="@EntityActionsColumn">

```

```

<DisplayTemplate>
    <EntityActions TItem="BookDto"
EntityActionsColumn="@EntityActionsColumn">
        <EntityAction TItem="BookDto"
            Text="@L["Edit"]"
            Clicked="() => OpenEditModalAsync(context)" />
    </EntityActions>
</DisplayTemplate>
</DataGridEntityActionsColumn>
```

```

\* `OpenEditModalAsync` is defined in the base class which takes the entity (book) to edit.

The `DataGridEntityActionsColumn` component is used to show an "Actions" dropdown for each row in the `DataGrid`. The `DataGridEntityActionsColumn` shows a \*\*single button\*\* instead of a dropdown if there is only one available action inside it:

![blazor-edit-book-action](./images/blazor-edit-book-action-3.png)

### ### Edit Modal

We can now define a modal to edit the book. Add the following code to the end of the `Books.razor` page:

```

```xml
<Modal @ref="@EditModal">
    <ModalBackdrop />
    <ModalContent IsCentered="true">
        <Form>
            <ModalHeader>
                <ModalTitle>@EditingEntity.Name</ModalTitle>
                <CloseButton Clicked="CloseEditModalAsync"/>
            </ModalHeader>
            <ModalBody>
                <Validations @ref="@EditValidationsRef" Model="@NewEntity"
ValidateOnLoad="false">
                    <Validation MessageLocalizer="@LH.Localize">
                        <Field>
                            <FieldLabel>@L["Name"]</FieldLabel>
                            <TextEdit @bind-Text="@EditingEntity.Name">
                                <Feedback>
                                    <ValidationError/>
                                </Feedback>
                            </TextEdit>
                        </Field>
                    </Validation>
                    <Field>
                        <FieldLabel>@L["Type"]</FieldLabel>
                        <Select TValue="BookType" @bind-
SelectedValue="@EditingEntity.Type">
                            @foreach (int bookTypeValue in
Enum.GetValues(typeof(BookType)))
                            {
                                <SelectItem TValue="BookType"
Value="@((BookType) bookTypeValue)">
                                    @L[$"Enum:BookType.{bookTypeValue}"]
                                </SelectItem>
                            }
                        </Select>
                    </Field>
                </Validations>
            </ModalBody>
        </Form>
    </ModalContent>
</Modal>
```

```

```

 </SelectItem>
 }
 </Select>
 </Field>
 <Field>
 <FieldLabel>@L["PublishDate"]</FieldLabel>
 <DateEdit TValue="DateTime" @bind-
Date="EditingEntity.PublishDate"/>
 </Field>
 <Field>
 <FieldLabel>@L["Price"]</FieldLabel>
 <NumericEdit TValue="float" @bind-
Value="EditingEntity.Price"/>
 </Field>
</Validations>
</ModalBody>
<ModalFooter>
 <Button Color="Color.Secondary"
 Clicked="CloseEditModalAsync">@L["Cancel"]</Button>
 <Button Color="Color.Primary"
 Type="@ButtonType.Submit"
 PreventDefaultOnSubmit="true"
 Clicked="UpdateEntityAsync">@L["Save"]</Button>
</ModalFooter>
</Form>
</ModalContent>
</Modal>
```

```

AutoMapper Configuration

The base `AbpCrudPageBase` uses the [object to object mapping](../Object-To-Object-Mapping.md) system to convert an incoming `BookDto` object to a `CreateUpdateBookDto` object. So, we need to define the mapping.

Open the `BookStoreBlazorAutoMapperProfile` inside the `Acme.BookStore.Blazor` project and change the content as the following:

```

```csharp
using Acme.BookStore.Books;
using AutoMapper;

namespace Acme.BookStore.Blazor;

public class BookStoreBlazorAutoMapperProfile : Profile
{
 public BookStoreBlazorAutoMapperProfile()
 {
 CreateMap<BookDto, CreateUpdateBookDto>();
 }
}
```

```

* We've just added the `CreateMap<BookDto, CreateUpdateBookDto>();` line to define the mapping.

Test the Editing Modal

You can now run the application and try to edit a book.

![blazor-edit-book-modal](./images/blazor-edit-book-modal-2.png)

> Tip: Try to leave the `*Name*` field empty and submit the form to show the validation error message.

Deleting a Book

Open the `'Books.razor'` page and add the following `'EntityAction'` code under the "Edit" action inside `'EntityActions'`:

```
```xml
<EntityAction TItem="BookDto"
 Text="@L["Delete"]"
 Clicked="() => DeleteEntityAsync(context)"
 ConfirmationMessage="() =>
GetDeleteConfirmationMessage(context)" />
````
```

- * `'DeleteEntityAsync'` is defined in the base class that deletes the entity by performing a call to the server.
- * `'ConfirmationMessage'` is a callback to show a confirmation message before executing the action.
- * `'GetDeleteConfirmationMessage'` is defined in the base class. You can override this method (or pass another value to the `'ConfirmationMessage'` parameter) to customize the localization message.

The "Actions" button becomes a dropdown since it has two actions now:

![blazor-delete-book-action](./images/blazor-delete-book-action-2.png)

Run the application and try to delete a book.

Full CRUD UI Code

Here's the complete code to create the book management CRUD page, that has been developed in the last two parts:

```
```xml
@page "/books"
@using Volo.Abp.Application.Dtos
@using Acme.BookStore.Books
@using Acme.BookStore.Localization
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Components.Web
@inject IStringLocalizer<BookStoreResource> L
@inject AbpBlazorMessageLocalizerHelper<BookStoreResource> LH
@inherits AbpCrudPageBase<IBookAppService, BookDto, Guid,
PagedAndSortedResultRequestDto, CreateUpdateBookDto>

<Card>
 <CardHeader>
 <Row Class="justify-content-between">
 <Column ColumnSize="ColumnSize.IsAuto">
 <h2>@L["Books"]</h2>
 </Column>
 <Column ColumnSize="ColumnSize.IsAuto">
```

```

 <Button Color="Color.Primary"
 Clicked="OpenCreateModalAsync">@L["NewBook"]</Button>
 </Column>
 </Row>
</CardHeader>
<CardBody>
 <DataGrid TItem="BookDto"
 Data="Entities"
 ReadData="OnDataGridReadAsync"
 CurrentPage="CurrentPage"
 TotalItems="TotalCount"
 ShowPager="true"
 PageSize="PageSize">
 <DataGridColumns>
 <DataGridEntityActionsColumn TItem="BookDto"
@ref="@EntityActionsColumn">
 <DisplayTemplate>
 <EntityActions TItem="BookDto"
EntityActionsColumn="@EntityActionsColumn">
 <EntityAction TItem="BookDto"
 Text="@L["Edit"]"
 Clicked="() =>
OpenEditModalAsync(context)" />
 <EntityAction TItem="BookDto"
 Text="@L["Delete"]"
 Clicked="() =>
DeleteEntityAsync(context)" />
 </EntityActions>
 </DisplayTemplate>
 </DataGridEntityActionsColumn>
 <DataGridColumn TItem="BookDto"
 Field="@nameof(BookDto.Name)"
 Caption="@L["Name"]"></DataGridColumn>
 <DataGridColumn TItem="BookDto"
 Field="@nameof(BookDto.Type)"
 Caption="@L["Type"]">
 <DisplayTemplate>
 @L[$"Enum:BookType.{context.Type}"]
 </DisplayTemplate>
 </DataGridColumn>
 <DataGridColumn TItem="BookDto"
 Field="@nameof(BookDto.PublishDate)"
 Caption="@L["PublishDate"]">
 <DisplayTemplate>
 @context.PublishDate.ToShortDateString()
 </DisplayTemplate>
 </DataGridColumn>
 <DataGridColumn TItem="BookDto"
 Field="@nameof(BookDto.Price)"
 Caption="@L["Price"]">
 </DataGridColumn>
 <DataGridColumn TItem="BookDto"
 Field="@nameof(BookDto.CreationTime)"
 Caption="@L["CreationTime"]">
 <DisplayTemplate>
 @context.CreationTime.ToString("yyyy-MM-dd")
 </DisplayTemplate>
 </DataGridColumn>
 </DataGridColumns>
 </DataGrid>

```

```

 </DisplayTemplate>
 </DataGridColumn>
 </DataGridColumns>
 </DataGrid>
</CardBody>
</Card>

<Modal @ref="@CreateModal">
 <ModalBackdrop />
 <ModalContent IsCentered="true">
 <Form>
 <ModalHeader>
 <ModalTitle>@L["NewBook"]</ModalTitle>
 <CloseButton Clicked="CloseCreateModalAsync"/>
 </ModalHeader>
 <ModalBody>
 <Validations @ref="@CreateValidationsRef" Model="@NewEntity"
ValidateOnLoad="false">
 <Validation MessageLocalizer="@LH.Localize">
 <Field>
 <FieldLabel>@L["Name"]</FieldLabel>
 <TextEdit @bind-Text="@NewEntity.Name">
 <Feedback>
 <ValidationError/>
 </Feedback>
 </TextEdit>
 </Field>
 </Validation>
 <Field>
 <FieldLabel>@L["Type"]</FieldLabel>
 <Select TValue="BookType" @bind-
SelectedValue="@NewEntity.Type">
 @foreach (int bookTypeValue in
Enum.GetValues(typeof(BookType)))
 {
 <SelectItem TValue="BookType"
Value="@((BookType) bookTypeValue)">
 @L[$"Enum:BookType.{bookTypeValue}"]
 </SelectItem>
 }
 </Select>
 </Field>
 <Field>
 <FieldLabel>@L["PublishDate"]</FieldLabel>
 <DateEdit TValue="DateTime" @bind-
Date="NewEntity.PublishDate"/>
 </Field>
 <Field>
 <FieldLabel>@L["Price"]</FieldLabel>
 <NumericEdit TValue="float" @bind-
Value="NewEntity.Price"/>
 </Field>
 </Validations>
 </ModalBody>
 <ModalFooter>
 <Button Color="Color.Secondary"
 Clicked="CloseCreateModalAsync">@L["Cancel"]</Button>
 <Button Color="Color.Primary"

```

```

 Type="@ButtonType.Submit"
 PreventDefaultOnSubmit="true"
 Clicked="CreateEntityAsync">@L["Save"]</Button>
 </ModalFooter>
</Form>
</ModalContent>
</Modal>

<Modal @ref="@EditModal">
 <ModalBackdrop />
 <ModalContent IsCentered="true">
 <Form>
 <ModalHeader>
 <ModalTitle>@EditingEntity.Name</ModalTitle>
 <CloseButton Clicked="CloseEditModalAsync"/>
 </ModalHeader>
 <ModalBody>
 <Validations @ref="@EditValidationsRef" Model="@NewEntity"
ValidateOnLoad="false">
 <Validation MessageLocalizer="@LH.Localize">
 <Field>
 <FieldLabel>@L["Name"]</FieldLabel>
 <TextEdit @bind-Text="@EditingEntity.Name">
 <Feedback>
 <ValidationError/>
 </Feedback>
 </TextEdit>
 </Field>
 </Validation>
 <Field>
 <FieldLabel>@L["Type"]</FieldLabel>
 <Select TValue="BookType" @bind-
SelectedValue="@EditingEntity.Type">
 @foreach (int bookTypeValue in
Enum.GetValues(typeof(BookType)))
 {
 <SelectItem TValue="BookType"
Value="@((BookType) bookTypeValue)">
 @L[$"Enum:BookType.{bookTypeValue}"]
 </SelectItem>
 }
 </Select>
 </Field>
 <Field>
 <FieldLabel>@L["PublishDate"]</FieldLabel>
 <DateEdit TValue="DateTime" @bind-
Date="EditingEntity.PublishDate"/>
 </Field>
 <Field>
 <FieldLabel>@L["Price"]</FieldLabel>
 <NumericEdit TValue="float" @bind-
Value="EditingEntity.Price"/>
 </Field>
 </Validations>
 </ModalBody>
 <ModalFooter>
 <Button Color="Color.Secondary"
Clicked="CloseEditModalAsync">@L["Cancel"]</Button>
 </ModalFooter>
 </Form>
 </ModalContent>
</Modal>

```

```

 <Button Color="Color.Primary"
 Type="@ButtonType.Submit"
 PreventDefaultOnSubmit="true"
 Clicked="UpdateEntityAsync">@L["Save"]</Button>
 </ModalFooter>
</Form>
</ModalContent>
</Modal>
```
{{end}}

```

The Next Part

Check out the [next part](Part-4.md) of this tutorial.

3.1.4 4: Integration Tests

```

# Web Application Development Tutorial - Part 4: Integration Tests
```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}
```

```

About This Tutorial

In this tutorial series, you will build an ABP based web application named 'Acme.BookStore'. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- * **{{DB_Value}}** as the ORM provider.
- * **{{UI_Value}}** as the UI Framework.

This tutorial is organized as the following parts;

- [Part 1: Creating the server side](Part-1.md)
- [Part 2: The book list page](Part-2.md)
- [Part 3: Creating, updating and deleting books](Part-3.md)
- **Part 4: Integration tests (this part)**
- [Part 5: Authorization](Part-5.md)
- [Part 6: Authors: Domain layer](Part-6.md)
- [Part 7: Authors: Database Integration](Part-7.md)
- [Part 8: Authors: Application Layer](Part-8.md)
- [Part 9: Authors: User Interface](Part-9.md)
- [Part 10: Book to Author Relation](Part-10.md)

Download the Source Code

This tutorial has multiple versions based on your **UI** and **Database** preferences. We've prepared a few combinations of the source code to be downloaded:

- * [MVC (Razor Pages) UI with EF Core](https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore)

```

* [Blazor UI with EF Core](https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore)
* [Angular UI with MongoDB](https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [this guide](../KB/Windows-Path-Too-Long-Fix.md).

{{if UI == "MVC" && DB == "EF"}}

```

Video Tutorial

This part is also recorded as a video tutorial and **published on YouTube**.

```

{{end}}

```

Test Projects in the Solution

This part covers the **server side** tests. There are several test projects in the solution:

```

![bookstore-test-projects-v2](./images/bookstore-test-projects-mvc.png)

> Test projects slightly differs based on your UI and Database selection. For example, if you select MongoDB, then the `Acme.BookStore.EntityFrameworkCore.Tests` will be `Acme.BookStore.MongoDB.Tests` .

```

Each project is used to test the related project. Test projects use the following libraries for testing:

```

* [Xunit](https://github.com/xunit/xunit) as the main test framework.
* [Shouldly](https://github.com/shouldly/shouldly) as the assertion library.
* [NSubstitute](http://nsubstitute.github.io/) as the mocking library.

{{if DB=="EF"}}

```

> The test projects are configured to use **SQLite in-memory** as the database. A separate database instance is created and seeded (with the [data seed system](../Data-Seeding.md)) to prepare a fresh database for every test.

```

{{else if DB=="Mongo"}}

```

> **[Mongo2Go](https://github.com/Mongo2Go/Mongo2Go)** library is used to mock the MongoDB database. A separate database instance is created and seeded (with the [data seed system](../Data-Seeding.md)) to prepare a fresh database for every test.

```

{{end}}

```

Adding Test Data

If you had created a data seed contributor as described in the [first part](Part-1.md), the same data will be available in your tests. So, you can skip this section. If you haven't created the seed contributor, you can use

the `BookStoreTestDataSeedContributor` to seed the same data to be used in the tests below.

Testing the BookAppService

Add a new test class, named `BookAppService_Tests` in the `Books` namespace (folder) of the `Acme.BookStore.Application.Tests` project:

```
```csharp
using System;
using System.Linq;
using System.Threading.Tasks;
using Shouldly;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Validation;
using Xunit;

namespace Acme.BookStore.Books;

{{if DB=="Mongo"}}
[Collection(BookStoreTestConsts.CollectionDefinitionName)]
{{end}}
public class BookAppService_Tests : BookStoreApplicationTestBase
{
 private readonly IBookAppService _bookAppService;

 public BookAppService_Tests()
 {
 _bookAppService = GetRequiredService<IBookAppService>();
 }

 [Fact]
 public async Task Should_Get_List_Of_Books()
 {
 //Act
 var result = await _bookAppService.GetListAsync(
 new PagedAndSortedResultRequestDto()
);

 //Assert
 result.TotalCount.ShouldBeGreaterThan(0);
 result.Items.ShouldContain(b => b.Name == "1984");
 }
}
```

```

* `Should_Get_List_Of_Books` test simply uses `BookAppService.GetListAsync` method to get and check the list of books.

* We can safely check the book "1984" by its name, because we know that this book is available in the database since we've added it in the seed data.

Add a new test method to the `BookAppService_Tests` class that creates a new **valid** book:

```
```csharp
[Fact]
public async Task Should_Create_A_Valid_Book()
{
```

```

//Act
var result = await _bookAppService.CreateAsync(
 new CreateUpdateBookDto
 {
 Name = "New test book 42",
 Price = 10,
 PublishDate = DateTime.Now,
 Type = BookType.ScienceFiction
 }
);

//Assert
result.Id.ShouldNotBe(Guid.Empty);
result.Name.ShouldBe("New test book 42");
}...

```

Add a new test that tries to create an invalid book and fails:

```

```csharp
[Fact]
public async Task Should_Not_Create_A_Book_Without_Name()
{
    var exception = await Assert.ThrowsAsync<AbpValidationException>(async () =>
    {
        await _bookAppService.CreateAsync(
            new CreateUpdateBookDto
            {
                Name = "",
                Price = 10,
                PublishDate = DateTime.Now,
                Type = BookType.ScienceFiction
            }
        );
    });

    exception.ValidationErrors
        .ShouldContain(err => err.MemberNames.Any(mem => mem == "Name"));
}...

```

* Since the `Name` is empty, ABP will throw an `AbpValidationException`.

The final test class should be as shown below:

```

```csharp
using System;
using System.Linq;
using System.Threading.Tasks;
using Shouldly;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Validation;
using Xunit;

namespace Acme.BookStore.Books;

{{if DB=="Mongo"}}

```

```

[Collection(BookStoreTestConsts.CollectionDefinitionName)]
{{end}}
public class BookAppService_Tests : BookStoreApplicationTestBase
{
 private readonly IBookAppService _bookAppService;

 public BookAppService_Tests()
 {
 _bookAppService = GetRequiredService<IBookAppService>();
 }

 [Fact]
 public async Task Should_Get_List_Of_Books()
 {
 //Act
 var result = await _bookAppService.GetListAsync(
 new PagedAndSortedResultRequestDto()
);

 //Assert
 result.TotalCount.ShouldBeGreaterThan(0);
 result.Items.ShouldContain(b => b.Name == "1984");
 }

 [Fact]
 public async Task Should_Create_A_Valid_Book()
 {
 //Act
 var result = await _bookAppService.CreateAsync(
 new CreateUpdateBookDto
 {
 Name = "New test book 42",
 Price = 10,
 PublishDate = DateTime.Now,
 Type = BookType.ScienceFiction
 }
);

 //Assert
 result.Id.ShouldNotBe(Guid.Empty);
 result.Name.ShouldBe("New test book 42");
 }

 [Fact]
 public async Task Should_Not_Create_A_Book_Without_Name()
 {
 var exception = await
Assert.ThrowsAsync<AbpValidationException>(async () =>
{
 await _bookAppService.CreateAsync(
 new CreateUpdateBookDto
 {
 Name = "",
 Price = 10,
 PublishDate = DateTime.Now,
 Type = BookType.ScienceFiction
 }
);
}

```

```

 });
 exception.ValidationErrors
 .ShouldContain(err => err.MemberNames.Any(mem => mem == "Name"));
}
}...

```

Open the \*\*Test Explorer Window\*\* (use Test -> Windows -> Test Explorer menu if it is not visible) and \*\*Run All\*\* tests:

```
![bookstore-appservice-tests](./images/bookstore-appservice-tests.png)
```

Congratulations, the \*\*green icons\*\* indicates that the tests have been successfully passed!

#### ## The Next Part

See the [\[next part\]](#)(Part-5.md) of this tutorial.

### 3.1.5 5: Authorization

```
Web Application Development Tutorial - Part 5: Authorization
```json
// [doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```

```

#### ## About This Tutorial

In this tutorial series, you will build an ABP based web application named `Acme.BookStore`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* \*\*{{DB\_Value}}\*\* as the ORM provider.
- \* \*\*{{UI\_Value}}\*\* as the UI Framework.

This tutorial is organized as the following parts;

- [\[Part 1: Creating the server side\]](#)(Part-1.md)
- [\[Part 2: The book list page\]](#)(Part-2.md)
- [\[Part 3: Creating, updating and deleting books\]](#)(Part-3.md)
- [\[Part 4: Integration tests\]](#)(Part-4.md)
- \*\*Part 5: Authorization (this part)\*\*
- [\[Part 6: Authors: Domain layer\]](#)(Part-6.md)
- [\[Part 7: Authors: Database Integration\]](#)(Part-7.md)
- [\[Part 8: Authors: Application Layer\]](#)(Part-8.md)
- [\[Part 9: Authors: User Interface\]](#)(Part-9.md)
- [\[Part 10: Book to Author Relation\]](#)(Part-10.md)

[### Download the Source Code](#)

This tutorial has multiple versions based on your \*\*UI\*\* and \*\*Database\*\* preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [MVC (Razor Pages) UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-EfCore>)
- \* [Blazor UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* [Angular UI with MongoDB](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [[this guide](#)](./KB/Windows-Path-Too-Long-Fix.md).

```
{{if UI == "MVC" && DB == "EF"}}
```

### ### Video Tutorial

This part is also recorded as a video tutorial and \*\*<a href="https://www.youtube.com/watch?v=1WsfMITN\_Jk&list=PLsNclT2aHJcPNaCf7Io3D bMN6yAk\_DgWJ&index=5" target="\_blank">published on YouTube</a>\*\*.

```
{{end}}
```

### ## Permissions

ABP Framework provides an [[authorization system](#)](./Authorization.md) based on the ASP.NET Core's [[authorization infrastructure](#)](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction>). One major feature added on top of the standard authorization infrastructure is the \*\*permission system\*\* which allows to define permissions and enable/disable per role, user or client.

### ### Permission Names

A permission must have a unique name (a `string`). The best way is to define it as a `const`, so we can reuse the permission name.

Open the `BookStorePermissions` class inside the `Acme.BookStore.Application.Contracts` project (in the `Permissions` folder) and change the content as shown below:

```
```csharp
namespace Acme.BookStore.Permissions;

public static class BookStorePermissions
{
    public const string GroupName = "BookStore";

    public static class Books
    {
        public const string Default = GroupName + ".Books";
        public const string Create = Default + ".Create";
        public const string Edit = Default + ".Edit";
        public const string Delete = Default + ".Delete";
    }
}
```

....

This is a hierarchical way of defining permission names. For example, "create book" permission name was defined as `BookStore.Books.Create`. ABP doesn't force you to a structure, but we find this way useful.

Permission Definitions

You should define permissions before using them.

Open the `BookStorePermissionDefinitionProvider` class inside the `Acme.BookStore.Application.Contracts` project (in the `Permissions` folder) and change the content as shown below:

```
```csharp
using Acme.BookStore.Localization;
using Volo.Abp.Authorization.Permissions;
using Volo.Abp.Localization;

namespace Acme.BookStore.Permissions;

public class BookStorePermissionDefinitionProvider :
PermissionDefinitionProvider
{
 public override void Define(IPermissionDefinitionContext context)
 {
 var bookStoreGroup = context.AddGroup(BookStorePermissions.GroupName,
L("Permission:BookStore"));

 var booksPermission =
bookStoreGroup.AddPermission(BookStorePermissions.Books.Default,
L("Permission:Books"));
 booksPermission.AddChild(BookStorePermissions.Books.Create,
L("Permission:Books.Create"));
 booksPermission.AddChild(BookStorePermissions.Books.Edit,
L("Permission:Books.Edit"));
 booksPermission.AddChild(BookStorePermissions.Books.Delete,
L("Permission:Books.Delete"));
 }

 private static LocalizableString L(string name)
 {
 return LocalizableString.Create<BookStoreResource>(name);
 }
}
````
```

This class defines a **permission group** (to group permissions on the UI, will be seen below) and **4 permissions** inside this group. Also, **Create**, **Edit** and **Delete** are children of the `BookStorePermissions.Books.Default` permission. A child permission can be selected **only if the parent was selected**.

Finally, edit the localization file (`en.json` under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project) to define the localization keys used above:

```
```json
```

```
"Permission:BookStore": "Book Store",
"Permission:Books": "Book Management",
"Permission:Books.Create": "Creating new books",
"Permission:Books.Edit": "Editing the books",
"Permission:Books.Delete": "Deleting the books"
```
```

> Localization key names are arbitrary and there is no forcing rule. But we prefer the convention used above.

Permission Management UI

Once you define the permissions, you can see them on the **permission management modal**.

Go to the *Administration -> Identity -> Roles* page, select *Permissions* action for the admin role to open the permission management modal:

![bookstore-permissions-ui](images/bookstore-permissions-ui-2.png)

Grant the permissions you want and save the modal.

> **Tip**: New permissions are automatically granted to the admin role if you run the `Acme.BookStore.DbMigrator` application.

Authorization

Now, you can use the permissions to authorize the book management.

Application Layer & HTTP API

Open the `BookAppService` class and set the policy names as the permission names defined above:

```
```csharp
using System;
using Acme.BookStore.Permissions;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Books;

public class BookAppService :
 CrudAppService<
 Book, //The Book entity
 BookDto, //Used to show books
 Guid, //Primary key of the book entity
 PagedAndSortedResultRequestDto, //Used for paging/sorting
 CreateUpdateBookDto>, //Used to create/update a book
 IBookAppService //implement the IBookAppService
{
 public BookAppService(IRepository<Book, Guid> repository)
 : base(repository)
 {
 GetPolicyName = BookStorePermissions.Books.Default;
 GetListPolicyName = BookStorePermissions.Books.Default;
 CreatePolicyName = BookStorePermissions.Books.Create;
 }
}
```

```

 UpdatePolicyName = BookStorePermissions.Books.Edit;
 DeletePolicyName = BookStorePermissions.Books.Delete;
 }
}
```

```

Added code to the constructor. Base `CrudAppService` automatically uses these permissions on the CRUD operations. This makes the **application service** secure, but also makes the **HTTP API** secure since this service is automatically used as an HTTP API as explained before (see [auto API controllers](../API/Auto-API-Controllers.md)).

> You will see the declarative authorization, using the `[Authorize(...)]` attribute, later while developing the author management functionality.

```
{{if UI == "MVC"}}

```

Razor Page

While securing the HTTP API & the application service prevents unauthorized users to use the services, they can still navigate to the book management page. While they will get authorization exception when the page makes the first AJAX call to the server, we should also authorize the page for a better user experience and security.

Open the `BookStoreWebModule` and add the following code block inside the `ConfigureServices` method:

```

```csharp
Configure<RazorPagesOptions>(options =>
{
 options.Conventions.AuthorizePage("/Books/Index",
BookStorePermissions.Books.Default);
 options.Conventions.AuthorizePage("/Books/CreateModal",
BookStorePermissions.Books.Create);
 options.Conventions.AuthorizePage("/Books/EditModal",
BookStorePermissions.Books.Edit);
});
```

```

Now, unauthorized users are redirected to the **login page**.

Hide the New Book Button

The book management page has a *New Book* button that should be invisible if the current user has no *Book Creation* permission.

![bookstore-new-book-button-small](images/bookstore-new-book-button-small-2.png)

Open the `Pages/Books/Index.cshtml` file and change the content as shown below:

```

```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Permissions
@using Acme.BookStore.Web.Pages.Books

```

```

@using Microsoft.AspNetCore.Authorization
@using Microsoft.Extensions.Localization
@model IndexModel
@inject IStringLocalizer<BookStoreResource> L
@inject IAuthorizationService AuthorizationService
@section scripts
{
 <abp-script src="/Pages/Books/Index.js"/>
}

<abp-card>
 <abp-card-header>
 <abp-row>
 <abp-column size-md="_6">
 <abp-card-title>@L["Books"]</abp-card-title>
 </abp-column>
 <abp-column size-md="_6" class="text-end">
 @if (await
AuthorizationService.IsGrantedAsync(BookStorePermissions.Books.Create))
 {
 <abp-button id="NewBookButton"
 text="@L["NewBook"].Value"
 icon="plus"
 button-type="Primary"/>
 }
 </abp-column>
 </abp-row>
 </abp-card-header>
 <abp-card-body>
 <abp-table striped-rows="true" id="BooksTable"></abp-table>
 </abp-card-body>
</abp-card>
```
* Added `@inject IAuthorizationService AuthorizationService` to access to the authorization service.
* Used `@if (await AuthorizationService.IsGrantedAsync(BookStorePermissions.Books.Create))` to check the book creation permission to conditionally render the *New Book* button.
```

JavaScript Side

Books table in the book management page has an actions button for each row. The actions button includes *Edit* and *Delete* actions:

```
![bookstore-edit-delete-actions](images/bookstore-edit-delete-actions-2.png)
```

We should hide an action if the current user has not granted for the related permission. Datatables row actions has a `visible` option that can be set to `false` to hide the action item.

Open the `Pages/Books/Index.js` inside the `Acme.BookStore.Web` project and add a `visible` option to the `Edit` action as shown below:

```
```js
{
 text: l('Edit'),
```

```

 visible: abp.auth.isGranted('BookStore.Books.Edit'), //CHECK for the
PERMISSION
 action: function (data) {
 editModal.open({ id: data.record.id });
 }
}...

```

Do same for the 'Delete' action:

```

```js
visible: abp.auth.isGranted('BookStore.Books.Delete')

* `abp.auth.isGranted(...)` is used to check a permission that is defined
before.
* `visible` could also be get a function that returns a `bool` if the value
will be calculated later, based on some conditions.

```

Menu Item

Even we have secured all the layers of the book management page, it is still visible on the main menu of the application. We should hide the menu item if the current user has no permission.

Open the 'BookStoreMenuContributor' class, find the code block below:

```

```csharp
context.Menu.AddItem(
 new ApplicationMenuItem(
 "BooksStore",
 l["Menu:BookStore"],
 icon: "fa fa-book"
).AddItem(
 new ApplicationMenuItem(
 "BooksStore.Books",
 l["Menu:Books"],
 url: "/Books"
)
)
);
```

```

And replace this code block with the following:

```

```csharp
context.Menu.AddItem(
 new ApplicationMenuItem(
 "BooksStore",
 l["Menu:BookStore"],
 icon: "fa fa-book"
).AddItem(
 new ApplicationMenuItem(
 "BooksStore.Books",
 l["Menu:Books"],
 url: "/Books"
).RequirePermissions(BookStorePermissions.Books.Default) // Check the
permission!
)
```

```

```
)  
}...  
)
```

We've only added the `' .RequirePermissions(BookStorePermissions.Books.Default)' extension method` call for the inner menu item.

```
 {{else if UI == "NG"}}

### Angular Guard Configuration
```

First step of the UI is to prevent unauthorized users to see the "Books" menu item and enter to the book management page.

Open the `'/src/app/book/book-routing.module.ts'` and replace with the following content:

```
```js
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AuthGuard, PermissionGuard } from '@abp/ng.core';
import { BookComponent } from './book.component';

const routes: Routes = [
 { path: '', component: BookComponent, canActivate: [AuthGuard, PermissionGuard] },
];

@NgModule({
 imports: [RouterModule.forChild(routes)],
 exports: [RouterModule],
})
export class BookRoutingModule {}
```

\* Imported `'AuthGuard'` and `'PermissionGuard'` from the `'@abp/ng.core'`.  
\* Added `'canActivate: [AuthGuard, PermissionGuard]'` to the route definition.

Open the `'/src/app/route.provider.ts'` and add `'requiredPolicy: 'BookStore.Books'` to the `'/books'` route. The `'/books'` route block should be following:

```
```js
{
  path: '/books',
  name: '::Menu:Books',
  parentName: '::Menu:BookStore',
  layout: eLayoutType.application,
  requiredPolicy: 'BookStore.Books',
}...  
```
```

```
Hide the New Book Button
```

The book management page has a `*New Book*` button that should be invisible if the current user has no `*Book Creation*` permission.

```
![bookstore-new-book-button-small](images/bookstore-new-book-button-small.png)
```

Open the `/src/app/book/book.component.html` file and replace the create button HTML content as shown below:

```
```html
<!-- Add the abpPermission directive -->
<button *abpPermission="'BookStore.Books.Create'" id="create" class="btn btn-primary" type="button" (click)="createBook()">
  <i class="fa fa-plus me-1"></i>
  <span>{{::NewBook | abpLocalization }}</span>
</button>
````
```

\* Just added `\*abpPermission="'BookStore.Books.Create'"` that hides the button if the current user has no permission.

### ### Hide the Edit and Delete Actions

Books table in the book management page has an actions button for each row. The actions button includes **Edit** and **Delete** actions:

```
![bookstore-edit-delete-actions](images/bookstore-edit-delete-actions-2.png)
```

We should hide an action if the current user has not granted for the related permission.

Open the `/src/app/book/book.component.html` file and replace the edit and delete buttons contents as shown below:

```
```html
<!-- Add the abpPermission directive -->
<button *abpPermission="'BookStore.Books.Edit'" ngbDropdownItem
(click)="editBook(row.id)"
  {{::Edit | abpLocalization }}>
</button>

<!-- Add the abpPermission directive -->
<button *abpPermission="'BookStore.Books.Delete'" ngbDropdownItem
(click)="delete(row.id)"
  {{::Delete | abpLocalization }}>
</button>
````
```

\* Added `\*abpPermission="'BookStore.Books.Edit'"` that hides the edit action if the current user has no editing permission.

\* Added `\*abpPermission="'BookStore.Books.Delete'"` that hides the delete action if the current user has no delete permission.

```
{{else if UI == "Blazor"}}
```

### ### Authorize the Razor Component

Open the `/Pages/Books.razor` file in the `Acme.BookStore.Blazor` project and add an **Authorize** attribute just after the **@page** directive and the following namespace imports (**@using** lines), as shown below:

```
```html
@page "/books"
@attribute [Authorize(BookStorePermissions.Books.Default)]
@using Acme.BookStore.Permissions
@using Microsoft.AspNetCore.Authorization
```

```

Adding this attribute prevents to enter this page if the current hasn't logged in or hasn't granted for the given permission. In case of attempt, the user is redirected to the login page.

#### ### Show/Hide the Actions

The book management page has a *\*New Book\** button and *\*Edit\** and *\*Delete\** actions for each book. We should hide these buttons/actions if the current user has not granted for the related permissions.

The base `'AbpCrudPageBase'` class already has the necessary functionality for these kind of operations.

#### #### Set the Policy (Permission) Names

Add the following code block to the end of the `'Books.razor'` file:

```
```csharp
@code
{
    public Books() // Constructor
    {
        CreatePolicyName = BookStorePermissions.Books.Create;
        UpdatePolicyName = BookStorePermissions.Books.Edit;
        DeletePolicyName = BookStorePermissions.Books.Delete;
    }
}
```

```

The base `'AbpCrudPageBase'` class automatically checks these permissions on the related operations. It also defines the given properties for us if we need to check them manually:

- \* `'HasCreatePermission'`: True, if the current user has permission to create the entity.
- \* `'HasUpdatePermission'`: True, if the current user has permission to edit/update the entity.
- \* `'HasDeletePermission'`: True, if the current user has permission to delete the entity.

> **\*\*Blazor Tip\*\*:** While adding the C# code into a `'@code'` block is fine for small code parts, it is suggested to use the code behind approach to develop a more maintainable code base when the code block becomes longer. We will use this approach for the authors part.

#### #### Hide the New Book Button

Wrap the *\*New Book\** button by an `'if'` block as shown below:

```
```xml
```

```

```

@if (HasCreatePermission)
{
 <Button Color="Color.Primary"
 Clicked="OpenCreateModalAsync">@L["NewBook"]</Button>
}
```
#### Hide the Edit/Delete Actions

`EntityAction` component defines `Visible` attribute (parameter) to conditionally show the action.
```

Update the `EntityActions` section as shown below:

```

```xml
<EntityActions TItem="BookDto" EntityActionsColumn="@EntityActionsColumn">
 <EntityAction TItem="BookDto"
 Text="@L["Edit"]"
 Visible=HasUpdatePermission
 Clicked="() => OpenEditModalAsync(context)" />
 <EntityAction TItem="BookDto"
 Text="@L["Delete"]"
 Visible=HasDeletePermission
 Clicked="() => DeleteEntityAsync(context)"

ConfirmationMessage="()=>GetDeleteConfirmationMessage(context)" />
</EntityActions>
```

```

About the Permission Caching

You can run and test the permissions. Remove a book related permission from the admin role to see the related button/action disappears from the UI.

****ABP Framework caches the permissions**** of the current user in the client side. So, when you change a permission for yourself, you need to manually ****refresh the page**** to take the effect. If you don't refresh and try to use the prohibited action you get an HTTP 403 (forbidden) response from the server.

> Changing a permission for a role or user immediately available on the server side. So, this cache system doesn't cause any security problem.

Menu Item

Even we have secured all the layers of the book management page, it is still visible on the main menu of the application. We should hide the menu item if the current user has no permission.

Open the `BookStoreMenuContributor` class in the `Acme.BookStore.Blazor` project, find the code block below:

```

```csharp
context.Menu.AddItem(
 new ApplicationMenuItem(
 "BooksStore",
 l["Menu:BookStore"],
 icon: "fa fa-book"
```

```

```

        ).AddItem(
            new ApplicationMenuItem(
                "BooksStore.Books",
                l["Menu:Books"],
                url: "/books"
            )
        );
    }
}
```

```

And replace this code block with the following:

```

```csharp
var bookStoreMenu = new ApplicationMenuItem(
    "BooksStore",
    l["Menu:BookStore"],
    icon: "fa fa-book"
);

context.Menu.AddItem(bookStoreMenu);

//CHECK the PERMISSION
if (await context.IsGrantedAsync(BookStorePermissions.Books.Default))
{
    bookStoreMenu.AddItem(new ApplicationMenuItem(
        "BooksStore.Books",
        l["Menu:Books"],
        url: "/books"
    ));
}
```

```

You also need to add `async` keyword to the `ConfigureMenuAsync` method and re-arrange the return value. The final `ConfigureMainMenuAsync` method should be the following:

```

```csharp
private async Task ConfigureMainMenuAsync(MenuConfigurationContext context)
{
    var l = context.GetLocalizer<BookStoreResource>();

    context.Menu.Items.Insert(
        0,
        new ApplicationMenuItem(
            "BookStore.Home",
            l["Menu:Home"],
            "/",
            icon: "fas fa-home"
        )
    );

    var bookStoreMenu = new ApplicationMenuItem(
        "BooksStore",
        l["Menu:BookStore"],
        icon: "fa fa-book"
    );

    context.Menu.AddItem(bookStoreMenu);
}
```

```

```

//CHECK the PERMISSION
if (await context.IsGrantedAsync(BookStorePermissions.Books.Default))
{
 bookStoreMenu.AddItem(new ApplicationMenuItem(
 "BooksStore.Books",
 l["Menu:Books"],
 url: "/books"
));
}
...
{{end}}

```

#### ## The Next Part

See the [\[next part\]](#)(Part-6.md) of this tutorial.

### 3.1.6 6: Authors: Domain layer

```

Web Application Development Tutorial - Part 6: Authors: Domain Layer
```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
...

```

About This Tutorial

In this tutorial series, you will build an ABP based web application named `'Acme.BookStore'`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- * **{{DB_Value}}** as the ORM provider.
- * **{{UI_Value}}** as the UI Framework.

This tutorial is organized as the following parts;

- [\[Part 1: Creating the server side\]](#)(Part-1.md)
- [\[Part 2: The book list page\]](#)(Part-2.md)
- [\[Part 3: Creating, updating and deleting books\]](#)(Part-3.md)
- [\[Part 4: Integration tests\]](#)(Part-4.md)
- [\[Part 5: Authorization\]](#)(Part-5.md)
- ****Part 6: Authors: Domain layer (this part)****
- [\[Part 7: Authors: Database Integration\]](#)(Part-7.md)
- [\[Part 8: Authors: Application Layer\]](#)(Part-8.md)
- [\[Part 9: Authors: User Interface\]](#)(Part-9.md)
- [\[Part 10: Book to Author Relation\]](#)(Part-10.md)

Download the Source Code

This tutorial has multiple versions based on your ****UI**** and ****Database**** preferences. We've prepared a few combinations of the source code to be downloaded:

- * [MVC (Razor Pages) UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- * [Blazor UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- * [Angular UI with MongoDB](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [this guide]([..KB/Windows-Path-Too-Long-Fix.md](#)).

Introduction

In the previous parts, we've used the ABP infrastructure to easily build some services;

- * Used the [CrudAppService]([../Application-Services.md](#)) base class instead of manually developing an application service for standard create, read, update and delete operations.
- * Used [generic repositories]([../Repositories.md](#)) to completely automate the database layer.

For the "Authors" part;

- * We will **do some of the things manually** to show how you can do it in case of need.
- * We will implement some **Domain Driven Design (DDD) best practices**.

> **The development will be done layer by layer to concentrate on an individual layer in one time. In a real project, you will develop your application feature by feature (vertical) as done in the previous parts. In this way, you will experience both approaches.**

The Author Entity

Create an `Authors` folder (namespace) in the `Acme.BookStore.Domain` project and add an `Author` class inside it:

```
```csharp
using System;
using JetBrains.Annotations;
using Volo.Abp;
using Volo.Abp.Domain.Entities.Auditing;

namespace Acme.BookStore.Authors;

public class Author : FullAuditedAggregateRoot<Guid>
{
 public string Name { get; private set; }
 public DateTime BirthDate { get; set; }
 public string ShortBio { get; set; }

 private Author()
 {
 /* This constructor is for deserialization / ORM purpose */
 }

 internal Author(
```

```

 Guid id,
 [NotNull] string name,
 DateTime birthDate,
 [CanBeNull] string shortBio = null)
 : base(id)
 {
 SetName(name);
 BirthDate = birthDate;
 ShortBio = shortBio;
 }

 internal Author ChangeName([NotNull] string name)
 {
 SetName(name);
 return this;
 }

 private void SetName([NotNull] string name)
 {
 Name = Check.NotNullOrWhiteSpace(
 name,
 nameof(name),
 maxLength: AuthorConsts.MaxNameLength
);
 }
}

```
```
* Inherited from `FullAuditedAggregateRoot<Guid>` which makes the entity
[soft delete](../Data-Filtering.md) (that means when you delete it, it is not
deleted in the database, but just marked as deleted) with all the
[auditing](../Entities.md) properties.
* `private set` for the `Name` property restricts to set this property from
out of this class. There are two ways of setting the name (in both cases, we
validate the name):
 * In the constructor, while creating a new author.
 * Using the `ChangeName` method to update the name later.
* The `constructor` and the `ChangeName` method is `internal` to force to use
these methods only in the domain layer, using the `AuthorManager` that will
be explained later.
* `Check` class is an ABP Framework utility class to help you while checking
method arguments (it throws `ArgumentException` on an invalid case).

`AuthorConsts` is a simple class that is located under the `Authors` namespace
(folder) of the `Acme.BookStore.Domain.Shared` project:

```

```

```csharp
namespace Acme.BookStore.Authors;

public static class AuthorConsts
{
    public const int MaxNameLength = 64;
}

```

```

Created this class inside the `Acme.BookStore.Domain.Shared` project since we will re-use it on the [Data Transfer Objects](../Data-Transfer-Objects.md) (DTOs) later.

#### ## AuthorManager: The Domain Service

'Author' constructor and 'ChangeName' methods are 'internal', so they can be used only in the domain layer. Create an 'AuthorManager' class in the 'Authors' folder (namespace) of the 'Acme.BookStore.Domain' project:

```
```csharp
using System;
using System.Threading.Tasks;
using JetBrains.Annotations;
using Volo.Abp;
using Volo.Abp.Domain.Services;

namespace Acme.BookStore.Authors;

public class AuthorManager : DomainService
{
    private readonly IAuthorRepository _authorRepository;

    public AuthorManager(IAuthorRepository authorRepository)
    {
        _authorRepository = authorRepository;
    }

    public async Task<Author> CreateAsync(
        [NotNull] string name,
        DateTime birthDate,
        [CanBeNull] string shortBio = null)
    {
        Check.NotNullOrWhiteSpace(name, nameof(name));

        var existingAuthor = await _authorRepository.FindByNameAsync(name);
        if (existingAuthor != null)
        {
            throw new AuthorAlreadyExistsException(name);
        }

        return new Author(
            GuidGenerator.Create(),
            name,
            birthDate,
            shortBio
        );
    }

    public async Task ChangeNameAsync(
        [NotNull] Author author,
        [NotNull] string newName)
    {
        Check.NotNull(author, nameof(author));
        Check.NotNullOrWhiteSpace(newName, nameof(newName));

        var existingAuthor = await
            _authorRepository.FindByNameAsync(newName);
```

```

        if (existingAuthor != null && existingAuthor.Id != author.Id)
    {
        throw new AuthorAlreadyExistsException(newName);
    }

    author.ChangeName(newName);
}
```
* 'AuthorManager' forces to create an author and change name of an author in a controlled way. The application layer (will be introduced later) will use these methods.

> DDD tip: Do not introduce domain service methods unless they are really needed and perform some core business rules. For this case, we needed this service to be able to force the unique name constraint.
```

Both methods checks if there is already an author with the given name and throws a special business exception, **'AuthorAlreadyExistsException'**, defined in the **'Acme.BookStore.Domain'** project (in the **'Authors'** folder) as shown below:

```

```csharp
using Volo.Abp;

namespace Acme.BookStore.Authors;

public class AuthorAlreadyExistsException : BusinessException
{
    public AuthorAlreadyExistsException(string name)
        : base(BookStoreDomainErrorCodes.AuthorAlreadyExists)
    {
        WithData("name", name);
    }
}
```

```

**'BusinessException'** is a special exception type. It is a good practice to throw domain related exceptions when needed. It is automatically handled by the ABP Framework and can be easily localized. **'WithData(...)'** method is used to provide additional data to the exception object that will later be used on the localization message or for some other purpose.

Open the **'BookStoreDomainErrorCodes'** in the **'Acme.BookStore.Domain.Shared'** project and change as shown below:

```

```csharp
namespace Acme.BookStore;

public static class BookStoreDomainErrorCodes
{
    public const string AuthorAlreadyExists = "BookStore:00001";
}
```

```

This is a unique string represents the error code thrown by your application and can be handled by client applications. For users, you probably want to

localize it. Open the `Localization/BookStore/en.json` inside the `Acme.BookStore.Domain.Shared` project and add the following entry:

```
```json
"BookStore:00001": "There is already an author with the same name: {name}"
```

```

Whenever you throw an `AuthorAlreadyExistsException`, the end user will see a nice error message on the UI.

## ## IAuthorRepository

`AuthorManager` injects the `IAuthorRepository`, so we need to define it. Create this new interface in the `Authors` folder (namespace) of the `Acme.BookStore.Domain` project:

```
```csharp
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Authors;

public interface IAuthorRepository : IRepository<Author, Guid>
{
    Task<Author> FindByNameAsync(string name);

    Task<List<Author>> GetListAsync(
        int skipCount,
        int maxResultCount,
        string sorting,
        string filter = null
    );
}

```


- `IAuthorRepository` extends the standard `IRepository<Author, Guid>` interface, so all the standard [repository](./Repositories.md) methods will also be available for the `IAuthorRepository`.
- `FindByNameAsync` was used in the `AuthorManager` to query an author by name.
- `GetListAsync` will be used in the application layer to get a listed, sorted and filtered list of authors to show on the UI.

```

We will implement this repository in the next part.

> Both of these methods might **seem unnecessary** since the standard repositories already provide generic querying methods and you can easily use them instead of defining such custom methods. You're right and do it like in a real application. However, for this **"learning"** tutorial, it is useful to explain how to create custom repository methods when you really need it.

## ## Conclusion

This part covered the domain layer of the authors functionality of the book store application. The main files created/updated in this part was highlighted in the picture below:

![bookstore-author-domain-layer](images/bookstore-author-domain-layer.png)

## ## The Next Part

See the [\[next part\]](#)(Part-7.md) of this tutorial.

### 3.1.7 7: Authors: Database Integration

```
Web Application Development Tutorial - Part 7: Authors: Database
Integration
```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```
About This Tutorial
```

In this tutorial series, you will build an ABP based web application named `Acme.BookStore`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* \*\*{{DB\_Value}}\*\* as the ORM provider.
- \* \*\*{{UI\_Value}}\*\* as the UI Framework.

This tutorial is organized as the following parts;

- [\[Part 1: Creating the server side\]](#)(Part-1.md)
- [\[Part 2: The book list page\]](#)(Part-2.md)
- [\[Part 3: Creating, updating and deleting books\]](#)(Part-3.md)
- [\[Part 4: Integration tests\]](#)(Part-4.md)
- [\[Part 5: Authorization\]](#)(Part-5.md)
- [\[Part 6: Authors: Domain layer\]](#)(Part-6.md)
- \*\*Part 7: Authors: Database Integration (this part)\*\*
- [\[Part 8: Authors: Application Layer\]](#)(Part-8.md)
- [\[Part 9: Authors: User Interface\]](#)(Part-9.md)
- [\[Part 10: Book to Author Relation\]](#)(Part-10.md)

## ### Download the Source Code

This tutorial has multiple versions based on your \*\*UI\*\* and \*\*Database\*\* preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [\[MVC \(Razor Pages\) UI with EF Core\]](#)(<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- \* [\[Blazor UI with EF Core\]](#)(<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* [\[Angular UI with MongoDB\]](#)(<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [\[this guide\]](#)(..../KB/Windows-Path-Too-Long-Fix.md).

```
Introduction
```

This part explains how to configure the database integration for the 'Author' entity introduced in the previous part.

```
{{if DB=="EF"}}
```

```
DB Context
```

Open the 'BookStoreDbContext' in the 'Acme.BookStore.EntityFrameworkCore' project and add the following 'DbSet' property:

```
```csharp
public DbSet<Author> Authors { get; set; }
````
```

Then locate to the 'OnModelCreating' method in 'BookStoreDbContext' class in the same project and add the following lines to the end of the method:

```
```csharp
builder.Entity<Author>(b =>
{
    b.ToTable(BookStoreConsts.DbTablePrefix + "Authors",
        BookStoreConsts.DbSchema);

    b.ConfigureByConvention();

    b.Property(x => x.Name)
        .IsRequired()
        .HasMaxLength(AuthorConsts.MaxNameLength);

    b.HasIndex(x => x.Name);
});
````
```

This is just like done for the 'Book' entity before, so no need to explain again.

```
Create a new Database Migration
```

The startup solution is configured to use [Entity Framework Core Code First Migrations](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). Since we've changed the database mapping configuration, we should create a new migration and apply changes to the database.

Open a command-line terminal in the directory of the 'Acme.BookStore.EntityFrameworkCore' project and type the following command:

```
```bash
dotnet ef migrations add Added_Authors
````
```

This will add a new migration class to the project:

![bookstore-efcore-migration-authors]("./images/bookstore-efcore-migration-authors.png")

You can apply changes to the database using the following command, in the same command-line terminal:

```
```bash
dotnet ef database update
```
```

> If you are using Visual Studio, you may want to use the `Add-Migration Added\_Authors` and `Update-Database` commands in the \*Package Manager Console (PMC)\*. In this case, ensure that `Acme.BookStore.EntityFrameworkCore` is the startup project in Visual Studio and `Acme.BookStore.EntityFrameworkCore` is the \*Default Project\* in PMC.

```
{{else if DB=="Mongo"}}
```

```
DB Context
```

Open the `BookStoreMongoDbContext` in the `MongoDb` folder of the `Acme.BookStore.MongoDB` project and add the following property to the class:

```
```csharp
public IMongoCollection<Author> Authors => Collection<Author>();
```
```

```
{{end}}
```

```
Implementing the IAuthorRepository
```

```
{{if DB=="EF"}}
```

Create a new class, named `EfCoreAuthorRepository` inside the `Acme.BookStore.EntityFrameworkCore` project (in the `Authors` folder) and paste the following code:

```
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Dynamic.Core;
using System.Threading.Tasks;
using Acme.BookStore.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using Volo.Abp.Domain.Repositories.EntityFrameworkCore;
using Volo.Abp.EntityFrameworkCore;

namespace Acme.BookStore.Authors;

public class EfCoreAuthorRepository
    : EfCoreRepository<BookStoreDbContext, Author, Guid>,
        IAuthorRepository
{
    public EfCoreAuthorRepository(
        IDbContextProvider<BookStoreDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }

    public async Task<Author> FindByNameAsync(string name)
```

```

    {
        var dbSet = await GetDbSetAsync();
        return await dbSet.FirstOrDefaultAsync(author => author.Name == name);
    }

    public async Task<List<Author>> GetListAsync(
        int skipCount,
        int maxResultCount,
        string sorting,
        string filter = null)
    {
        var dbSet = await GetDbSetAsync();
        return await dbSet
            .WhereIf(
                !filter.IsNullOrWhiteSpace(),
                author => author.Name.Contains(filter)
            )
            .OrderBy(sorting)
            .Skip(skipCount)
            .Take(maxResultCount)
            .ToListAsync();
    }
}

```


* Inherited from the `EfCoreRepository`, so it inherits the standard repository method implementations.

* `WhereIf` is a shortcut extension method of the ABP Framework. It adds the `Where` condition only if the first condition meets (it filters by name, only if the filter was provided). You could do the same yourself, but these type of shortcut methods makes our life easier.

* `sorting` can be a string like `Name`, `Name ASC` or `Name DESC`. It is possible by using the \[System.Linq.Dynamic.Core\]\(https://www.nuget.org/packages/System.Linq.Dynamic.Core\) NuGet package.

> See the \[EF Core Integration document\]\(../Entity-Framework-Core.md\) for more information on the EF Core based repositories.


```

`{{else if DB=="Mongo"}}`

Create a new class, named `MongoDBAuthorRepository` inside the `Acme.BookStore.MongoDB` project (in the `Authors` folder) and paste the following code:

```

```csharp
using System;
using System.Linq;
using System.Linq.Dynamic.Core;
using System.Collections.Generic;
using System.Threading.Tasks;
using Acme.BookStore.MongoDB;
using MongoDB.Driver;
using MongoDB.Driver.Linq;
using Volo.Abp.Domain.Repositories.MongoDB;
using Volo.Abp.MongoDB;
```

```

namespace Acme.BookStore.Authors;

public class MongoDbAuthorRepository
    : MongoDbRepository<BookStoreMongoDbContext, Author, Guid>,
    IAuthorRepository
{
    public MongoDbAuthorRepository(
        IMongoDbContextProvider<BookStoreMongoDbContext> dbContextProvider
    ) : base(dbContextProvider)
    {
    }

    public async Task<Author> FindByNameAsync(string name)
    {
        var queryable = await GetMongoQueryableAsync();
        return await queryable.FirstOrDefaultAsync(author => author.Name == name);
    }

    public async Task<List<Author>> GetListAsync(
        int skipCount,
        int maxResultCount,
        string sorting,
        string filter = null)
    {
        var queryable = await GetMongoQueryableAsync();
        return await queryable
            .WhereIf<Author, IMongoQueryable<Author>>(
                !filter.IsNullOrWhiteSpace(),
                author => author.Name.Contains(filter))
            .OrderBy(sorting)
            .As<IMongoQueryable<Author>>()
            .Skip(skipCount)
            .Take(maxResultCount)
            .ToListAsync();
    }
}
```

```

- \* Inherited from the `MongoDbRepository`, so it inherits the standard repository method implementations.
- \* `WhereIf` is a shortcut extension method of the ABP Framework. It adds the `Where` condition only if the first condition meets (it filters by name, only if the filter was provided). You could do the same yourself, but these type of shortcut methods makes our life easier.
- \* `sorting` can be a string like `Name`, `Name ASC` or `Name DESC`. It is possible by using the [\[System.Linq.Dynamic.Core\]\(https://www.nuget.org/packages/System.Linq.Dynamic.Core\)](https://www.nuget.org/packages/System.Linq.Dynamic.Core) NuGet package.

> See the [\[MongoDB Integration document\]\(../MongoDB.md\)](#) for more information on the MongoDB based repositories.

`{{end}}`

`## The Next Part`

See the [\[next part\]](#)(Part-8.md) of this tutorial.

### 3.1.8 8: Authors: Application Layer

```
Web Application Development Tutorial - Part 8: Authors: Application Layer
```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```
About This Tutorial
```

In this tutorial series, you will build an ABP based web application named '[`Acme.BookStore`](#)'. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* \*\*{{DB\_Value}}\*\* as the ORM provider.
- \* \*\*{{UI\_Value}}\*\* as the UI Framework.

This tutorial is organized as the following parts;

- [\[Part 1: Creating the server side\]](#)(Part-1.md)
- [\[Part 2: The book list page\]](#)(Part-2.md)
- [\[Part 3: Creating, updating and deleting books\]](#)(Part-3.md)
- [\[Part 4: Integration tests\]](#)(Part-4.md)
- [\[Part 5: Authorization\]](#)(Part-5.md)
- [\[Part 6: Authors: Domain layer\]](#)(Part-6.md)
- [\[Part 7: Authors: Database Integration\]](#)(Part-7.md)
- \*\*Part 8: Author: Application Layer (this part)\*\*
- [\[Part 9: Authors: User Interface\]](#)(Part-9.md)
- [\[Part 10: Book to Author Relation\]](#)(Part-10.md)

#### ### Download the Source Code

This tutorial has multiple versions based on your \*\*UI\*\* and \*\*Database\*\* preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [\[MVC \(Razor Pages\) UI with EF Core\]](#)(<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- \* [\[Blazor UI with EF Core\]](#)(<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* [\[Angular UI with MongoDB\]](#)(<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [\[this guide\]](#)(../KB/Windows-Path-Too-Long-Fix.md).

#### ## Introduction

This part explains to create an application layer for the '[`Author`](#)' entity created before.

#### ## IAuthorAppService

We will first create the [application service](../Application-Services.md) interface and the related [DTO](../Data-Transfer-Objects.md)s. Create a new interface, named `IAuthorAppService`, in the `Authors` namespace (folder) of the `Acme.BookStore.Application.Contracts` project:

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;

namespace Acme.BookStore.Authors;

public interface IAuthorAppService : IApplicationService
{
    Task<AuthorDto> GetAsync(Guid id);

    Task<PagedResultDto<AuthorDto>> GetListAsync(GetAuthorListDto input);

    Task<AuthorDto> CreateAsync(CreateAuthorDto input);

    Task UpdateAsync(Guid id, UpdateAuthorDto input);

    Task DeleteAsync(Guid id);
}
```

```

- \* `IApplicationService` is a conventional interface that is inherited by all the application services, so the ABP Framework can identify the service.
- \* Defined standard methods to perform CRUD operations on the `Author` entity.
- \* `PagedResultDto` is a pre-defined DTO class in the ABP Framework. It has an `Items` collection and a `TotalCount` property to return a paged result.
- \* Preferred to return an `AuthorDto` (for the newly created author) from the `CreateAsync` method, while it is not used by this application – just to show a different usage.

This interface is using the DTOs defined below (create them for your project).

```
AuthorDto

```csharp
using System;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Authors;

public class AuthorDto : EntityDto<Guid>
{
    public string Name { get; set; }

    public DateTime BirthDate { get; set; }

    public string ShortBio { get; set; }
}
```

```

\* `EntityDto<T>` simply has an `Id` property with the given generic argument.  
You could create an `Id` property yourself instead of inheriting the `EntityDto<T>`.

### ### GetAuthorListDto

```
```csharp
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Authors;

public class GetAuthorListDto : PagedAndSortedResultRequestDto
{
    public string? Filter { get; set; }
}...```

```

* `Filter` is used to search authors. It can be `null` (or empty string) to get all the authors.

* `PagedAndSortedResultRequestDto` has the standard paging and sorting properties: `int MaxResultCount`, `int SkipCount` and `string Sorting`.

> ABP Framework has such base DTO classes to simplify and standardize your DTOs. See the [\[DTO documentation\]](#)(../Data-Transfer-Objects.md) for all.

CreateAuthorDto

```
```csharp
using System;
using System.ComponentModel.DataAnnotations;

namespace Acme.BookStore.Authors;

public class CreateAuthorDto
{
 [Required]
 [StringLength(AuthorConsts.MaxNameLength)]
 public string Name { get; set; }

 [Required]
 public DateTime BirthDate { get; set; }

 public string ShortBio { get; set; }
}...```

```

Data annotation attributes can be used to validate the DTO. See the [\[validation document\]](#)(../Validation.md) for details.

### ### UpdateAuthorDto

```
```csharp
using System;
using System.ComponentModel.DataAnnotations;

namespace Acme.BookStore.Authors;

public class UpdateAuthorDto

```

```

{
    [Required]
    [StringLength(AuthorConsts.MaxNameLength)]
    public string Name { get; set; }

    [Required]
    public DateTime BirthDate { get; set; }

    public string ShortBio { get; set; }
}
...

```

> We could share (re-use) the same DTO among the create and the update operations. While you can do it, we prefer to create different DTOs for these operations since we see they generally be different by the time. So, code duplication is reasonable here compared to a tightly coupled design.

AuthorAppService

It is time to implement the `IAuthorAppService` interface. Create a new class, named `AuthorAppService` in the `Authors` namespace (folder) of the `Acme.BookStore.Application` project:

```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Permissions;
using Microsoft.AspNetCore.Authorization;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Authors;

[Authorize(BookStorePermissions.Authors.Default)]
public class AuthorAppService : BookStoreAppService, IAuthorAppService
{
 private readonly IAuthorRepository _authorRepository;
 private readonly AuthorManager _authorManager;

 public AuthorAppService(
 IAuthorRepository authorRepository,
 AuthorManager authorManager)
 {
 _authorRepository = authorRepository;
 _authorManager = authorManager;
 }

 //...SERVICE METHODS WILL COME HERE...
}
...

```

\* `'[Authorize(BookStorePermissions.Authors.Default)]'` is a declarative way to check a permission (policy) to authorize the current user. See the [\[authorization document\]\(../Authorization.md\)](#) for more.  
`BookStorePermissions` class will be updated below, don't worry for the compile error for now.

- \* Derived from the `BookStoreAppService`, which is a simple base class comes with the startup template. It is derived from the standard `ApplicationService` class.
- \* Implemented the `IAuthorAppService` which was defined above.
- \* Injected the `IAuthorRepository` and `AuthorManager` to use in the service methods.

Now, we will introduce the service methods one by one. Copy the explained method into the `AuthorAppService` class.

```
GetAsync
```

```
```csharp
public async Task<AuthorDto> GetAsync(Guid id)
{
    var author = await _authorRepository.GetAsync(id);
    return ObjectMapper.Map<Author, AuthorDto>(author);
}
...```

```

This method simply gets the `Author` entity by its `Id`, converts to the `AuthorDto` using the [object to object mapper](../Object-To-Object-Mapping.md). This requires to configure the AutoMapper, which will be explained later.

```
### GetListAsync
```

```
```csharp
public async Task<PagedResultDto<AuthorDto>> GetListAsync(GetAuthorListDto input)
{
 if (input.Sorting.IsNullOrWhiteSpace())
 {
 input.Sorting = nameof(Author.Name);
 }

 var authors = await _authorRepository.GetListAsync(
 input.SkipCount,
 input.MaxResultCount,
 input.Sorting,
 input.Filter
);

 var totalCount = input.Filter == null
 ? await _authorRepository.CountAsync()
 : await _authorRepository.CountAsync(
 author => author.Name.Contains(input.Filter));

 return new PagedResultDto<AuthorDto>(
 totalCount,
 ObjectMapper.Map<List<Author>, List<AuthorDto>>(authors)
);
}
...```

```

\* Default sorting is "by author name" which is done in the beginning of the method in case of it wasn't sent by the client.

- \* Used the `IAuthorRepository.GetListAsync` to get a paged, sorted and filtered list of authors from the database. We had implemented it in the previous part of this tutorial. Again, it actually was not needed to create such a method since we could directly query over the repository, but wanted to demonstrate how to create custom repository methods.
- \* Directly queried from the `AuthorRepository` while getting the count of the authors. If a filter is sent, then we are using it to filter entities while getting the count.
- \* Finally, returning a paged result by mapping the list of `Author`s to a list of `AuthorDto`s.

### ### CreateAsync

```
```csharp
[Authorize(BookStorePermissions.Authors.Create)]
public async Task<AuthorDto> CreateAsync(CreateAuthorDto input)
{
    var author = await _authorManager.CreateAsync(
        input.Name,
        input.BirthDate,
        input.ShortBio
    );

    await _authorRepository.InsertAsync(author);

    return ObjectMapper.Map<Author, AuthorDto>(author);
}
```

```

- \* `CreateAsync` requires the `BookStorePermissions.Authors.Create` permission (in addition to the `BookStorePermissions.Authors.Default` declared for the `AuthorAppService` class).
- \* Used the `AuthorManager` (domain service) to create a new author.
- \* Used the `IAuthorRepository.InsertAsync` to insert the new author to the database.
- \* Used the `ObjectMapper` to return an `AuthorDto` representing the newly created author.

> \*\*DDD tip\*\*: Some developers may find useful to insert the new entity inside the `AuthorManager.CreateAsync`. We think it is a better design to leave it to the application layer since it better knows when to insert it to the database (maybe it requires additional works on the entity before insert, which would require to an additional update if we perform the insert in the domain service). However, it is completely up to you.

### ### UpdateAsync

```
```csharp
[Authorize(BookStorePermissions.Authors.Edit)]
public async Task UpdateAsync(Guid id, UpdateAuthorDto input)
{
    var author = await _authorRepository.GetAsync(id);

    if (author.Name != input.Name)
    {
        await _authorManager.ChangeNameAsync(author, input.Name);
    }
}
```

```

```

 author.BirthDate = input.BirthDate;
 author.ShortBio = input.ShortBio;

 await _authorRepository.UpdateAsync(author);
 }
}

* `UpdateAsync` requires the additional `BookStorePermissions.Authors.Edit` permission.
* Used the `IAuthorRepository.GetAsync` to get the author entity from the database. `GetAsync` throws `EntityNotFoundException` if there is no author with the given id, which results a `404` HTTP status code in a web application. It is a good practice to always bring the entity on an update operation.
* Used the `AuthorManager.ChangeNameAsync` (domain service method) to change the author name if it was requested to change by the client.
* Directly updated the `BirthDate` and `ShortBio` since there is not any business rule to change these properties, they accept any value.
* Finally, called the `IAuthorRepository.UpdateAsync` method to update the entity on the database.

{{if DB == "EF"}}
 > **EF Core Tip**: Entity Framework Core has a **change tracking** system and **automatically saves** any change to an entity at the end of the unit of work (You can simply think that the ABP Framework automatically calls `SaveChanges` at the end of the method). So, it will work as expected even if you don't call the `._authorRepository.UpdateAsync(...)` in the end of the method. If you don't consider to change the EF Core later, you can just remove this line.

{{end}}
DeleteAsync
```csharp
[Authorize(BookStorePermissions.Authors.Delete)]
public async Task DeleteAsync(Guid id)
{
    await _authorRepository.DeleteAsync(id);
}
```
* `DeleteAsync` requires the additional `BookStorePermissions.Authors.Delete` permission.
* It simply uses the `DeleteAsync` method of the repository.

Permission Definitions
```

You can't compile the code since it is expecting some constants declared in the `BookStorePermissions` class.

Open the `BookStorePermissions` class inside the `Acme.BookStore.Application.Contracts` project (in the `Permissions` folder) and change the content as shown below:

```

```csharp
namespace Acme.BookStore.Permissions;
```

```

public static class BookStorePermissions
{
    public const string GroupName = "BookStore";

    public static class Books
    {
        public const string Default = GroupName + ".Books";
        public const string Create = Default + ".Create";
        public const string Edit = Default + ".Edit";
        public const string Delete = Default + ".Delete";
    }

    // *** ADDED a NEW NESTED CLASS ***
    public static class Authors
    {
        public const string Default = GroupName + ".Authors";
        public const string Create = Default + ".Create";
        public const string Edit = Default + ".Edit";
        public const string Delete = Default + ".Delete";
    }
}
...

```

Then open the `BookStorePermissionDefinitionProvider` in the same project and add the following lines at the end of the `Define` method:

```

```csharp
var authorsPermission = bookStoreGroup.AddPermission(
 BookStorePermissions.Authors.Default, L("Permission:Authors"));
authorsPermission.AddChild(
 BookStorePermissions.Authors.Create, L("Permission:Authors.Create"));
authorsPermission.AddChild(
 BookStorePermissions.Authors.Edit, L("Permission:Authors.Edit"));
authorsPermission.AddChild(
 BookStorePermissions.Authors.Delete, L("Permission:Authors.Delete"));
```

```

Finally, add the following entries to the `Localization/BookStore/en.json` inside the `Acme.BookStore.Domain.Shared` project, to localize the permission names:

```

```csharp
"Permission:Authors": "Author Management",
"Permission:Authors.Create": "Creating new authors",
"Permission:Authors.Edit": "Editing the authors",
"Permission:Authors.Delete": "Deleting the authors"
```

```

Object to Object Mapping

The `AuthorAppService` is using the `ObjectMapper` to convert the `Author` objects to `AuthorDto` objects. So, we need to define this mapping in the AutoMapper configuration.

Open the `BookStoreApplicationAutoMapperProfile` class inside the `Acme.BookStore.Application` project and add the following line to the constructor:

```
```csharp
CreateMap<Author, AuthorDto>();
```
```

Data Seeder

As just done for the books before, it would be good to have some initial author entities in the database. This will be good while running the application first time, but also it is very useful for the automated tests.

Open the `'BookStoreDataSeederContributor'` in the `'Acme.BookStore.Domain'` project and change the file content with the code below:

```
```csharp
using System;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Books;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore;

public class BookStoreDataSeederContributor
 : IDataSeedContributor, ITransientDependency
{
 private readonly IRepository<Book, Guid> _bookRepository;
 private readonly IAuthorRepository _authorRepository;
 private readonly AuthorManager _authorManager;

 public BookStoreDataSeederContributor(
 IRepository<Book, Guid> bookRepository,
 IAuthorRepository authorRepository,
 AuthorManager authorManager)
 {
 _bookRepository = bookRepository;
 _authorRepository = authorRepository;
 _authorManager = authorManager;
 }

 public async Task SeedAsync(DataSeedContext context)
 {
 if (await _bookRepository.GetCountAsync() <= 0)
 {
 await _bookRepository.InsertAsync(
 new Book
 {
 Name = "1984",
 Type = BookType.Dystopia,
 PublishDate = new DateTime(1949, 6, 8),
 Price = 19.84f
 },
 autoSave: true
);

 await _bookRepository.InsertAsync(

```

```

 new Book
 {
 Name = "The Hitchhiker's Guide to the Galaxy",
 Type = BookType.ScienceFiction,
 PublishDate = new DateTime(1995, 9, 27),
 Price = 42.0f
 },
 autoSave: true
);
}

// ADDED SEED DATA FOR AUTHORS

if (await _authorRepository.GetCountAsync() <= 0)
{
 await _authorRepository.InsertAsync(
 await _authorManager.CreateAsync(
 "George Orwell",
 new DateTime(1903, 06, 25),
 "Orwell produced literary criticism and poetry, fiction and polemical journalism; and is best known for the allegorical novella Animal Farm (1945) and the dystopian novel Nineteen Eighty-Four (1949)."
)
);

 await _authorRepository.InsertAsync(
 await _authorManager.CreateAsync(
 "Douglas Adams",
 new DateTime(1952, 03, 11),
 "Douglas Adams was an English author, screenwriter, essayist, humorist, satirist and dramatist. Adams was an advocate for environmentalism and conservation, a lover of fast cars, technological innovation and the Apple Macintosh, and a self-proclaimed 'radical atheist'."
)
);
}
}
```

```

{{if DB=="EF"}}

You can now run the `DbMigrator` console application to **migrate** the **database schema** and **seed** the initial data.

{{else if DB=="Mongo"}}

You can now run the `DbMigrator` console application to **seed** the initial data.

{{end}}

Testing the Author Application Service

Finally, we can write some tests for the `IAuthorAppService`. Add a new class, named `AuthorAppService_Tests` in the `Authors` namespace (folder) of the `Acme.BookStore.Application.Tests` project:

```

```csharp
using System;
using System.Threading.Tasks;
using Shouldly;
using Xunit;

namespace Acme.BookStore.Authors;

{{if DB=="Mongo"}}
[Collection(BookStoreTestConsts.CollectionDefinitionName)]
{{end}}
public class AuthorAppService_Tests : BookStoreApplicationTestBase
{
 private readonly IAuthorAppService _authorAppService;

 public AuthorAppService_Tests()
 {
 _authorAppService = GetRequiredService<IAuthorAppService>();
 }

 [Fact]
 public async Task Should_Get_All_Authors_Without_Any_Filter()
 {
 var result = await _authorAppService.GetListAsync(new
GetAuthorListDto());

 result.TotalCount.ShouldBeGreaterThanOrEqualTo(2);
 result.Items.ShouldContain(author => author.Name == "George Orwell");
 result.Items.ShouldContain(author => author.Name == "Douglas Adams");
 }

 [Fact]
 public async Task Should_Get_Filtered_Authors()
 {
 var result = await _authorAppService.GetListAsync(
 new GetAuthorListDto {Filter = "George"});

 result.TotalCount.ShouldBeGreaterThanOrEqualTo(1);
 result.Items.ShouldContain(author => author.Name == "George Orwell");
 result.Items.ShouldNotContain(author => author.Name == "Douglas
Adams");
 }

 [Fact]
 public async Task Should_Create_A_New_Author()
 {
 var authorDto = await _authorAppService.CreateAsync(
 new CreateAuthorDto
 {
 Name = "Edward Bellamy",
 BirthDate = new DateTime(1850, 05, 22),
 ShortBio = "Edward Bellamy was an American author..."
 });
 authorDto.Id.ShouldNotBe(Guid.Empty);
 authorDto.Name.ShouldBe("Edward Bellamy");
 }
}

```

```

[Fact]
public async Task Should_Not_Allow_To_Create_Duplicate_Author()
{
 await Assert.ThrowsAsync<AuthorAlreadyExistsException>(async () =>
 {
 await _authorAppService.CreateAsync(
 new CreateAuthorDto
 {
 Name = "Douglas Adams",
 BirthDate = DateTime.Now,
 ShortBio = "..."
 }
);
 });
}

//TODO: Test other methods...
}...

```

Created some tests for the application service methods, which should be clear to understand.

## ## The Next Part

See the [\[next part\]](#)(Part-9.md) of this tutorial.

### 3.1.9 9: Authors: User Interface

```

Web Application Development Tutorial - Part 9: Authors: User Interface
```json
//[doc-params]
{
    "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
    "DB": ["EF", "Mongo"]
}
```

```

## ## About This Tutorial

In this tutorial series, you will build an ABP based web application named `'Acme.BookStore'`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- \* \*\*{{DB\_Value}}\*\* as the ORM provider.
- \* \*\*{{UI\_Value}}\*\* as the UI Framework.

This tutorial is organized as the following parts;

- [\[Part 1: Creating the server side\]](#)(Part-1.md)
- [\[Part 2: The book list page\]](#)(Part-2.md)
- [\[Part 3: Creating, updating and deleting books\]](#)(Part-3.md)
- [\[Part 4: Integration tests\]](#)(Part-4.md)
- [\[Part 5: Authorization\]](#)(Part-5.md)
- [\[Part 6: Authors: Domain layer\]](#)(Part-6.md)
- [\[Part 7: Authors: Database Integration\]](#)(Part-7.md)

- [Part 8: Authors: Application Layer](Part-8.md)
- \*\*Part 9: Authors: User Interface (this part)\*\*
- [Part 10: Book to Author Relation](Part-10.md)

### ### Download the Source Code

This tutorial has multiple versions based on your \*\*UI\*\* and \*\*Database\*\* preferences. We've prepared a few combinations of the source code to be downloaded:

- \* [MVC (Razor Pages) UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- \* [Blazor UI with EF Core](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* [Angular UI with MongoDB](<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [this guide](../KB/Windows-Path-Too-Long-Fix.md).

### ## Introduction

This part explains how to create a CRUD page for the `Author` entity introduced in the previous parts.

```
{{if UI == "MVC"}}
The Authors List Page
```

Create a new razor page, `Index.cshtml` under the `Pages/Authors` folder of the `Acme.BookStore.Web` project and change the content as given below.

```
Index.cshtml

```
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Permissions
@using Acme.BookStore.Web.Pages.Authors
@using Microsoft.AspNetCore.Authorization
@using Microsoft.Extensions.Localization
@inject IStringLocalizer<BookStoreResource> L
@inject IAuthorizationService AuthorizationService
@model IndexModel

@section scripts
{
    <abp-script src="/Pages/Authors/Index.js"/>
}

<abp-card>
    <abp-card-header>
        <abp-row>
            <abp-column size-md="_6">
                <abp-card-title>@L["Authors"]</abp-card-title>
            </abp-column>
            <abp-column size-md="_6" class="text-end">
                @if (await AuthorizationService
```

```

        .IsGrantedAsync(BookStorePermissions.Authors.Create))
    {
        <abp-button id="NewAuthorButton"
            text="@L["NewAuthor"].Value"
            icon="plus"
            button-type="Primary"/>
    }
</abp-column>
</abp-row>
</abp-card-header>
<abp-card-body>
    <abp-table striped-rows="true" id="AuthorsTable"></abp-table>
</abp-card-body>
</abp-card>
```

```

This is a simple page similar to the Books page we had created before. It imports a JavaScript file which will be introduced below.

```

Index.cshtml.cs

```csharp
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace Acme.BookStore.Web.Pages.Authors;

public class IndexModel : PageModel
{
    public void OnGet()
    {

    }
}
```

Index.js

```js
$(function () {
    var l = abp.localization.getResource('BookStore');
    var createModal = new abp.ModalManager(abp.appPath +
'Authors/CreateModal');
    var editModal = new abp.ModalManager(abp.appPath + 'Authors/EditModal');

    var dataTable = $('#AuthorsTable').DataTable(
        abp.libs.datatables.normalizeConfiguration({
            serverSide: true,
            paging: true,
            order: [[1, "asc"]],
            searching: false,
            scrollX: true,
            ajax:
                abp.libs.datatables.createAjax(acme.bookStore.authors.author.getList),
            columnDefs: [
                {
                    title: l('Actions'),
                    rowAction: {
                        items:
```

```

```

 [
 {
 text: l('Edit'),
 visible:
 abp.auth.isGranted('BookStore.Authors.Edit'),
 action: function (data) {
 editModal.open({ id:
 data.record.id });
 }
 },
 {
 text: l('Delete'),
 visible:
 abp.auth.isGranted('BookStore.Authors.Delete'),
 confirmMessage: function (data) {
 return l(
 'AuthorDeletionConfirmationMessage',
 data.record.name
);
 },
 action: function (data) {
 acme.bookStore.authors.author
 .delete(data.record.id)
 .then(function() {
 abp.notify.info(
 l('SuccessfullyDeleted')
);
 dataTable.ajax.reload();
 });
 }
 }
]
 },
 {
 title: l('Name'),
 data: "name"
 },
 {
 title: l('BirthDate'),
 data: "birthDate",
 render: function (data) {
 return luxon
 .DateTime
 .fromISO(data, {
 locale: abp.localization.currentCulture.name
 }).toLocaleString();
 }
 }
]
);
createModal.onResult(function () {
 dataTable.ajax.reload();
}
);

```

```

 });

 editModal.onResult(function () {
 dataTable.ajax.reload();
 });

 $('#NewAuthorButton').click(function (e) {
 e.preventDefault();
 createModal.open();
 });
});

```

Briefly, this JavaScript page;

- \* Creates a Data table with 'Actions', 'Name' and 'BirthDate' columns.
- \* 'Actions' column is used to add \*Edit\* and \*Delete\* actions.
- \* 'BirthDate' provides a 'render' function to format the 'DateTime' value using the [luxon](<https://moment.github.io/luxon/>) library.
- \* Uses the `app.ModalManager` to open \*Create\* and \*Edit\* modal forms.

This code is very similar to the Books page created before, so we will not explain it more.

### ### Localizations

This page uses some localization keys we need to declare. Open the 'en.json' file under the 'Localization/BookStore' folder of the 'Acme.BookStore.Domain.Shared' project and add the following entries:

```

```json
"Menu:Authors": "Authors",
"Authors": "Authors",
"AuthorDeletionConfirmationMessage": "Are you sure to delete the author
'{0}'?",
"BirthDate": "Birth date",
"NewAuthor": "New author"
```

```

Notice that we've added more keys. They will be used in the next sections.

### ### Add to the Main Menu

Open the 'BookStoreMenuContributor.cs' in the 'Menus' folder of the 'Acme.BookStore.Web' project and add a new \*Authors\* menu item under the \*Book Store\* menu item. The following code (in the 'ConfigureMainMenuAsync' method) shows the final code part:

```

```csharp
context.Menu.AddItem(
    new ApplicationMenuItem(
        "BooksStore",
        l["Menu:BookStore"],
        icon: "fa fa-book"
    ).AddItem(
        new ApplicationMenuItem(
            "BooksStore.Books",
            l["Menu:Books"],

```

```

        url: "/Books"
    ).RequirePermissions(BookStorePermissions.Books.Default)
).AddItem( // ADDED THE NEW "AUTHORS" MENU ITEM UNDER THE "BOOK STORE"
MENU
    new ApplicationMenuItem(
        "BooksStore.Authors",
        l["Menu:Authors"],
        url: "/Authors"
    ).RequirePermissions(BookStorePermissions.Authors.Default)
)
);
...

```

Run the Application

Run and login to the application. **You can not see the menu item since you don't have permission yet.** Go to the `Identity/Roles` page, click to the *Actions* button and select the *Permissions* action for the **admin role**:

![bookstore-author-permissions](images/bookstore-author-permissions-3.png)

As you see, the admin role has no *Author Management* permissions yet. Click to the checkboxes and save the modal to grant the necessary permissions. You will see the *Authors* menu item under the *Book Store* in the main menu, after **refreshing the page**:

![bookstore-authors-page](images/bookstore-authors-page-3.png)

The page is fully working except *New author* and *Actions/Edit* since we haven't implemented them yet.

> **Tip**: If you run the `DbMigrator` console application after defining a new permission, it automatically grants these new permissions to the admin role and you don't need to manually grant the permissions yourself.

Create Modal

Create a new razor page, `CreateModal.cshtml` under the `Pages/Authors` folder of the `Acme.BookStore.Web` project and change the content as given below.

CreateModal.cshtml

```

```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Authors
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Modal
@model CreateModalModel
@inject IStringLocalizer<BookStoreResource> L
 @{
 Layout = null;
}
<form asp-page="/Authors/CreateModal">
 <abp-modal>
 <abp-modal-header title="@L["NewAuthor"].Value"></abp-modal-header>
 <abp-modal-body>

```

```

 <abp-input asp-for="Author.Name" />
 <abp-input asp-for="Author.BirthDate" />
 <abp-input asp-for="Author.ShortBio" />
 </abp-modal-body>
 <abp-modal-footer
buttons="@{AbpModalButtons.Cancel|AbpModalButtons.Save)"></abp-modal-footer>
</abp-modal>
</form>
```

```

We had used [dynamic forms](../UI/AspNetCore/Tag-Helpers/Dynamic-Forms.md) of the ABP Framework for the books page before. We could use the same approach here, but we wanted to show how to do it manually. Actually, not so manually, because we've used `abp-input` tag helper in this case to simplify creating the form elements.

You can definitely use the standard Bootstrap HTML structure, but it requires to write a lot of code. `abp-input` automatically adds validation, localization and other standard elements based on the data type.

CreateModal.cshtml.cs

```

```csharp
using System;
using System.ComponentModel.DataAnnotations;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Form;

namespace Acme.BookStore.Web.Pages.Authors;

public class CreateModalModel : BookStorePageModel
{
 [BindProperty]
 public CreateAuthorViewModel Author { get; set; }

 private readonly IAuthorAppService _authorAppService;

 public CreateModalModel(IAuthorAppService authorAppService)
 {
 _authorAppService = authorAppService;
 }

 public void OnGet()
 {
 Author = new CreateAuthorViewModel();
 }

 public async Task<IActionResult> OnPostAsync()
 {
 var dto = ObjectMapper.Map<CreateAuthorViewModel,
CreateAuthorDto>(Author);
 await _authorAppService.CreateAsync(dto);
 return NoContent();
 }

 public class CreateAuthorViewModel

```

```

 {
 [Required]
 [StringLength(AuthorConsts.MaxNameLength)]
 public string Name { get; set; }

 [Required]
 [DataType(DataType.Date)]
 public DateTime BirthDate { get; set; }

 [TextArea]
 public string ShortBio { get; set; }
 }
}..

```

This page model class simply injects and uses the `IAuthorAppService` to create a new author. The main difference between the book creation model class is that this one is declaring a new class, `CreateAuthorViewModel`, for the view model instead of re-using the `CreateAuthorDto`.

The main reason of this decision was to show you how to use a different model class inside the page. But there is one more benefit: We added two attributes to the class members, which were not present in the `CreateAuthorDto`:

- \* Added `[DataType(DataType.Date)]` attribute to the `BirthDate` which shows a date picker on the UI for this property.
- \* Added `[TextArea]` attribute to the `ShortBio` which shows a multi-line text area instead of a standard textbox.

In this way, you can specialize the view model class based on your UI requirements without touching to the DTO. As a result of this decision, we have used `ObjectMapper` to map `CreateAuthorViewModel` to `CreateAuthorDto`. To be able to do that, you need to add a new mapping code to the `BookStoreWebAutoMapperProfile` constructor:

```

```csharp
using Acme.BookStore.Authors; // ADDED NAMESPACE IMPORT
using Acme.BookStore.Books;
using AutoMapper;

namespace Acme.BookStore.Web;

public class BookStoreWebAutoMapperProfile : Profile
{
    public BookStoreWebAutoMapperProfile()
    {
        CreateMap<BookDto, CreateUpdateBookDto>();

        // ADD A NEW MAPPING
        CreateMap<Pages.Authors.CreateModalModel.CreateAuthorViewModel,
                  CreateAuthorDto>();
    }
}...

```

"New author" button will work as expected and open a new model when you run the application again:

```

![bookstore-new-author-modal](images/bookstore-new-author-modal-2.png)

## Edit Modal

Create a new razor page, `EditModal.cshtml` under the `Pages/Authors` folder of the `Acme.BookStore.Web` project and change the content as given below.

### EditModal.cshtml

```html
```
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Authors
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Modal
@model EditModalModel
@inject IStringLocalizer<BookStoreResource> L
 @{
     Layout = null;
 }
<form asp-page="/Authors/EditModal">
    <abp-modal>
        <abp-modal-header title="@L["Update"].Value"></abp-modal-header>
        <abp-modal-body>
            <abp-input asp-for="Author.Id" />
            <abp-input asp-for="Author.Name" />
            <abp-input asp-for="Author.BirthDate" />
            <abp-input asp-for="Author.ShortBio" />
        </abp-modal-body>
        <abp-modal-footer>
            buttons="@((AbpModalButtons.Cancel | AbpModalButtons.Save))" </abp-modal-footer>
        </abp-modal>
    </form>
```

EditModal.cshtml.cs

```csharp
```
using System;
using System.ComponentModel.DataAnnotations;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Form;

namespace Acme.BookStore.Web.Pages.Authors;

public class EditModalModel : BookStorePageModel
{
 [BindProperty]
 public EditAuthorViewModel Author { get; set; }

 private readonly IAuthorAppService _authorAppService;

 public EditModalModel(IAuthorAppService authorAppService)
 {
 _authorAppService = authorAppService;
 }
}
```

```

```

public async Task OnGetAsync(Guid id)
{
    var authorDto = await _authorAppService.GetAsync(id);
    Author = ObjectMapper.Map<AuthorDto, EditAuthorViewModel>(authorDto);
}

public async Task<IActionResult> OnPostAsync()
{
    await _authorAppService.UpdateAsync(
        Author.Id,
        ObjectMapper.Map<EditAuthorViewModel, UpdateAuthorDto>(Author)
    );

    return NoContent();
}

public class EditAuthorViewModel
{
    [HiddenInput]
    public Guid Id { get; set; }

    [Required]
    [StringLength(AuthorConsts.MaxNameLength)]
    public string Name { get; set; }

    [Required]
    [DataType(DataType.Date)]
    public DateTime BirthDate { get; set; }

    [TextArea]
    public string ShortBio { get; set; }
}
```

```

This class is similar to the `CreateModal.cshtml.cs` while there are some main differences;

- \* Uses the `IAuthorAppService.GetAsync(...)` method to get the editing author from the application layer.
- \* `EditAuthorViewModel` has an additional `Id` property which is marked with the `[HiddenInput]` attribute that creates a hidden input for this property.

This class requires to add two object mapping declarations to the `BookStoreWebAutoMapperProfile` class:

```

```csharp
using Acme.BookStore.Authors;
using Acme.BookStore.Books;
using AutoMapper;

namespace Acme.BookStore.Web;

public class BookStoreWebAutoMapperProfile : Profile
{
    public BookStoreWebAutoMapperProfile()
    {

```

```

        CreateMap<BookDto, CreateUpdateBookDto>();

        CreateMap<Pages.Authors.CreateModalModel.CreateAuthorViewModel,
                  CreateAuthorDto>();

        // ADD THESE NEW MAPPINGS
        CreateMap<AuthorDto,
Pages.Authors.EditModalModel.EditAuthorViewModel>();
        CreateMap<Pages.Authors.EditModalModel.EditAuthorViewModel,
                  UpdateAuthorDto>();
    }
}
```

```

That's all! You can run the application and try to edit an author.

```

{{else if UI == "NG"}}
The Author Management Page

```

Run the following command line to create a new module, named `AuthorModule` in the root folder of the angular application:

```

```bash
yarn ng generate module author --module app --routing --route authors
```

```

This command should produce the following output:

```

```bash
> yarn ng generate module author --module app --routing --route authors

yarn run v1.19.1
$ ng generate module author --module app --routing --route authors
CREATE src/app/author/author-routing.module.ts (344 bytes)
CREATE src/app/author/author.module.ts (349 bytes)
CREATE src/app/author/author.component.html (21 bytes)
CREATE src/app/author/author.component.spec.ts (628 bytes)
CREATE src/app/author/author.component.ts (276 bytes)
CREATE src/app/author/author.component.scss (0 bytes)
UPDATE src/app/app-routing.module.ts (1396 bytes)
Done in 2.22s.
```

```

### AuthorModule

Open the `/src/app/author/author.module.ts` and replace the content as shown below:

```

```js
import { NgModule } from '@angular/core';
import { SharedModule } from '../shared/shared.module';
import { AuthorRoutingModule } from './author-routing.module';
import { AuthorComponent } from './author.component';
import { NgbDatepickerModule } from '@ng-bootstrap/ng-bootstrap';

@NgModule({
  declarations: [AuthorComponent],

```

```
    imports: [SharedModule, AuthorRoutingModule, NgbDatepickerModule],
})
export class AuthorModule {}  
```

```

- Added the ` SharedModule`. ` SharedModule` exports some common modules needed to create user interfaces.
- ` SharedModule` already exports the ` CommonModule`, so we've removed the ` CommonModule`.
- Added ` NgbDatepickerModule` that will be used later on the author create and edit forms.

### ### Menu Definition

Open the `src/app/route.provider.ts` file and add the following menu definition:

```
```js
{
  path: '/authors',
  name: '::Menu:Authors',
  parentName: '::Menu:BookStore',
  layout: eLayoutType.application,
  requiredPolicy: 'BookStore.Authors',
}
```

```

The final `configureRoutes` function declaration should be following:

```
```js
function configureRoutes(routes: RoutesService) {
  return () => {
    routes.add([
      {
        path: '/',
        name: '::Menu:Home',
        iconClass: 'fas fa-home',
        order: 1,
        layout: eLayoutType.application,
      },
      {
        path: '/book-store',
        name: '::Menu:BookStore',
        iconClass: 'fas fa-book',
        order: 2,
        layout: eLayoutType.application,
      },
      {
        path: '/books',
        name: '::Menu:Books',
        parentName: '::Menu:BookStore',
        layout: eLayoutType.application,
        requiredPolicy: 'BookStore.Books',
      },
      {
        path: '/authors',
        name: '::Menu:Authors',
        parentName: '::Menu:BookStore',
      }
    ])
  }
}
```

```

```

 layout: eLayoutType.application,
 requiredPolicy: 'BookStore.Authors',
 },
],
);
}
```

```

Service Proxy Generation

[ABP CLI](<https://docs.abp.io/en/abp/latest/CLI>) provides `generate-proxy` command that generates client proxies for your HTTP APIs to make easy to consume your HTTP APIs from the client side. Before running `generate-proxy` command, your host must be up and running.

Run the following command in the `angular` folder:

```
```bash
abp generate-proxy -t ng
```

```

This command generates the service proxy for the author service and the related model (DTO) classes:

![bookstore-angular-service-proxy-author](images/bookstore-angular-service-proxy-author-2.png)

AuthorComponent

Open the `/src/app/author/author.component.ts` file and replace the content as below:

```

```js
import { Component, OnInit } from '@angular/core';
import { ListService, PagedResultDto } from '@abp/ng.core';
import { AuthorService, AuthorDto } from '@proxy/authors';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { NgbDateNativeAdapter, NgbDateAdapter } from '@ng-bootstrap/ng-bootstrap';
import { ConfirmationService, Confirmation } from '@abp/ng.theme.shared';

@Component({
 selector: 'app-author',
 templateUrl: './author.component.html',
 styleUrls: ['./author.component.scss'],
 providers: [ListService, { provide: NgbDateAdapter, useClass: NgbDateNativeAdapter }],
})
export class AuthorComponent implements OnInit {
 author = { items: [], totalCount: 0 } as PagedResultDto<AuthorDto>;
 isModalOpen = false;
 form: FormGroup;
 selectedAuthor = {} as AuthorDto;
}

constructor()
```

```

```
public readonly list: ListService,
private authorService: AuthorService,
private fb: FormBuilder,
private confirmation: ConfirmationService
) {}

ngOnInit(): void {
  const authorStreamCreator = (query) => this.authorService.getList(query);

  this.list.hookToQuery(authorStreamCreator).subscribe((response) => {
    this.author = response;
  });
}

createAuthor() {
  this.selectedAuthor = {} as AuthorDto;
  this.buildForm();
  this.isModalOpen = true;
}

editAuthor(id: string) {
  this.authorService.get(id).subscribe((author) => {
    this.selectedAuthor = author;
    this.buildForm();
    this.isModalOpen = true;
  });
}

buildForm() {
  this.form = this.fb.group({
    name: [this.selectedAuthor.name || '', Validators.required],
    birthDate: [
      this.selectedAuthor.birthDate ? new
      Date(this.selectedAuthor.birthDate) : null,
      Validators.required,
    ],
  });
}

save() {
  if (this.form.invalid) {
    return;
  }

  if (this.selectedAuthor.id) {
    this.authorService
      .update(this.selectedAuthor.id, this.form.value)
      .subscribe(() => {
        this.isModalOpen = false;
        this.form.reset();
        this.list.get();
      });
  } else {
    this.authorService.create(this.form.value).subscribe(() => {
      this.isModalOpen = false;
      this.form.reset();
      this.list.get();
    });
  }
}
```

```

        }
    }

    delete(id: string) {
        this.confirmation.warn('::AreYouSureToDelete', '::AreYouSure')
            .subscribe((status) => {
                if (status === Confirmation.Status.confirm) {
                    this.authorService.delete(id).subscribe(() => this.list.get());
                }
            });
    }
}
```

```

Open the `/src/app/author/author.component.html` and replace the content as below:

```

```html
<div class="card">
    <div class="card-header">
        <div class="row">
            <div class="col col-md-6">
                <h5 class="card-title">
                    {::Menu:Authors' | abpLocalization }}}

```

```

                <button *abpPermission="'BookStore.Authors.Edit'" ngbDropdownItem (click)="editAuthor(row.id)">
                    {%{{ 'Edit' | abpLocalization }}%}
                </button>
                <button *abpPermission="'BookStore.Authors.Delete'" ngbDropdownItem (click)="delete(row.id)">
                    {%{{ 'Delete' | abpLocalization }}%}
                </button>
            </div>
        </div>
    </ng-template>
</ngx-datatable-column>
<ngx-datatable-column [name]="'::Name' | abpLocalization" prop="name"></ngx-datatable-column>
<ngx-datatable-column [name]="'::BirthDate' | abpLocalization">
    <ng-template let-row="row" ngx-datatable-cell-template>
        {%{{ row.birthDate | date }}%}
    </ng-template>
</ngx-datatable-column>
</ngx-datatable>
</div>
</div>

<abp-modal [(visible)]="isModalOpen">
    <ng-template #abpHeader>
        <h3>{{selectedAuthor.id ? '::Edit' : '::NewAuthor'} | abpLocalization }}</h3>
    </ng-template>

    <ng-template #abpBody>
        <form [formGroup]="form" (ngSubmit)="save()">
            <div class="form-group">
                <label for="author-name">Name</label><span> * </span>
                <input type="text" id="author-name" class="form-control" formControlName="name" autofocus />
            </div>

            <div class="mt-2">
                <label>Birth date</label><span> * </span>
                <input #datepicker="ngbDatepicker" class="form-control" name="datepicker" formControlName="birthDate" ngbDatepicker (click)="datepicker.toggle()"/>
            </div>
        </form>
    </ng-template>

    <ng-template #abpFooter>
        <button type="button" class="btn btn-secondary" abpClose>
            {%{{ 'Close' | abpLocalization }}%}
        </button>

        <button class="btn btn-primary" (click)="save()" [disabled]="form.invalid">

```

```
<i class="fa fa-check mr-1"></i>
{{'::Save' | abpLocalization}}
</button>
</ng-template>
</abp-modal>
````
```

### ### Localizations

This page uses some localization keys we need to declare. Open the `en.json` file under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project and add the following entries:

```
```json
"Menu:Authors": "Authors",
"Authors": "Authors",
"AuthorDeletionConfirmationMessage": "Are you sure to delete the author
'{0}'?",
"BirthDate": "Birth date",
"NewAuthor": "New author"
```
```

### ### Run the Application

Run and login to the application. \*\*You can not see the menu item since you don't have permission yet.\*\* Go to the `identity/roles` page, click to the \*Actions\* button and select the \*Permissions\* action for the \*\*admin role\*\*:

![bookstore-author-permissions](images/bookstore-author-permissions-2.png)

As you see, the admin role has no \*Author Management\* permissions yet. Click to the checkboxes and save the modal to grant the necessary permissions. You will see the \*Authors\* menu item under the \*Book Store\* in the main menu, after \*\*refreshing the page\*\*:

![bookstore-authors-page](images/bookstore-angular-authors-page-2.png)

That's all! This is a fully working CRUD page, you can create, edit and delete authors.

> \*\*Tip\*\*: If you run the `DbMigrator` console application after defining a new permission, it automatically grants these new permissions to the admin role and you don't need to manually grant the permissions yourself.

```
 {{end}}
{{if UI == "Blazor" || UI == "BlazorServer"}}
The Author Management Page
```

### ### Authors Razor Component

Create a new Razor Component Page, `/Pages/Authors.razor`, in the `Acme.BookStore.Blazor` project with the following content:

```
```xml
@page "/authors"
@using Acme.BookStore.Authors
```

```

@using Acme.BookStore.Localization
@using Volo.Abp.AspNetCore.Components.Web
@inherits BookStoreComponentBase
@inject IAuthorAppService AuthorAppService
@inject AbpBlazorMessageLocalizerHelper<BookStoreResource> LH
<Card>
    <CardHeader>
        <Row Class="justify-content-between">
            <Column ColumnSize="ColumnSize.IsAuto">
                <h2>@L["Authors"]</h2>
            </Column>
            <Column ColumnSize="ColumnSize.IsAuto">
                @if (CanCreateAuthor)
                {
                    <Button Color="Color.Primary"
                            Clicked="OpenCreateAuthorModal">
                        @L["NewAuthor"]
                    </Button>
                }
            </Column>
        </Row>
    </CardHeader>
    <CardBody>
        <DataGrid TItem="AuthorDto"
                  Data="AuthorList"
                  ReadData="OnDataGridReadAsync"
                  TotalItems="TotalCount"
                  ShowPager="true"
                  PageSize="PageSize">
            <DataGridColumns>
                <DataGridColumn Width="150px"
                                TItem="AuthorDto"
                                Field="@nameof(AuthorDto.Id)"
                                Sortable="false"
                                Caption="@L["Actions"]">
                    <DisplayTemplate>
                        <Dropdown>
                            <DropdownToggle Color="Color.Primary">
                                @L["Actions"]
                            </DropdownToggle>
                            <DropdownMenu>
                                @if (CanEditAuthor)
                                {
                                    <DropdownItem Clicked="() =>
OpenEditAuthorModal(context)">
                                        @L["Edit"]
                                    </DropdownItem>
                                }
                                @if (CanDeleteAuthor)
                                {
                                    <DropdownItem Clicked="() =>
DeleteAuthorAsync(context)">
                                        @L["Delete"]
                                    </DropdownItem>
                                }
                            </DropdownMenu>
                        </Dropdown>
                    </DisplayTemplate>
                </DataGridColumn>
            </DataGridColumns>
        </DataGrid>
    </CardBody>
</Card>

```



```

        @L["Cancel"]
    </Button>
    <Button Color="Color.Primary"
        Type="@ButtonType.Submit"
        PreventDefaultOnSubmit="true"
        Clicked="CreateAuthorAsync">
        @L["Save"]
    </Button>
</ModalFooter>
</Form>
</ModalContent>
</Modal>

<Modal @ref="EditAuthorModal">
    <ModalBackdrop />
    <ModalContent IsCentered="true">
        <Form>
            <ModalHeader>
                <ModalTitle>@EditingAuthor.Name</ModalTitle>
                <CloseButton Clicked="CloseEditAuthorModal" />
            </ModalHeader>
            <ModalBody>
                <Validations @ref="@EditValidationsRef"
Model="@EditingAuthor" ValidateOnLoad="false">
                    <Validation MessageLocalizer="@LH.Localize">
                        <Field>
                            <FieldLabel>@L["Name"]</FieldLabel>
                            <TextEdit @bind-Text="@EditingAuthor.Name">
                                <Feedback>
                                    <ValidationError/>
                                </Feedback>
                            </TextEdit>
                        </Field>
                    </Validation>
                    <Field>
                        <FieldLabel>@L["BirthDate"]</FieldLabel>
                        <DatePicker TValue="DateTime" @bind-
Date="@EditingAuthor.BirthDate"/>
                    </Field>
                    <Validation>
                        <Field>
                            <FieldLabel>@L["ShortBio"]</FieldLabel>
                            <MemoEdit Rows="5" @bind-
Text="@EditingAuthor.ShortBio">
                                <Feedback>
                                    <ValidationError/>
                                </Feedback>
                            </MemoEdit>
                        </Field>
                    </Validation>
                </Validations>
            </ModalBody>
            <ModalFooter>
                <Button Color="Color.Secondary"
                    Clicked="CloseEditAuthorModal">
                    @L["Cancel"]
                </Button>
                <Button Color="Color.Primary">

```

```

        Type="@ButtonType.Submit"
        PreventDefaultOnSubmit="true"
        Clicked="UpdateAuthorAsync">
            @L["Save"]
        </Button>
    </ModalFooter>
</Form>
</ModalContent>
</Modal>
````

* This code is similar to the `Books.razor`, except it doesn't inherit from the `AbpCrudPageBase`, but uses its own implementation.
* Injects the `IAuthorAppService` to consume the server side HTTP APIs from the UI. We can directly inject application service interfaces and use just like regular method calls by the help of [Dynamic C# HTTP API Client Proxy System](../API/Dynamic-CSharp-API-Clients.md), which performs REST API calls for us. See the `Authors` class below to see the usage.
* Injects the `IAuthorizationService` to check [permissions](../Authorization.md).
* Injects the `IObjectMapper` for [object to object mapping](../Object-To-Object-Mapping.md).
```

Create a new code behind file, `Authors.razor.cs`, under the `Pages` folder, with the following content:

```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Permissions;
using Blazorise;
using Blazorise.DataGrid;
using Microsoft.AspNetCore.Authorization;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Blazor.Pages;

public partial class Authors
{
    private IReadOnlyList<AuthorDto> AuthorList { get; set; }

    private int PageSize { get; } =
    LimitedResultRequestDto.DefaultMaxResultCount;
    private int CurrentPage { get; set; }
    private string CurrentSorting { get; set; }
    private int TotalCount { get; set; }

    private bool CanCreateAuthor { get; set; }
    private bool CanEditAuthor { get; set; }
    private bool CanDeleteAuthor { get; set; }

    private CreateAuthorDto NewAuthor { get; set; }

    private Guid EditingAuthorId { get; set; }
    private UpdateAuthorDto EditingAuthor { get; set; }
}
```

```

private Modal CreateAuthorModal { get; set; }
private Modal EditAuthorModal { get; set; }

private Validations CreateValidationsRef;
private Validations EditValidationsRef;

public Authors()
{
    NewAuthor = new CreateAuthorDto();
    EditingAuthor = new UpdateAuthorDto();
}

protected override async Task OnInitializedAsync()
{
    await SetPermissionsAsync();
    await GetAuthorsAsync();
}

private async Task SetPermissionsAsync()
{
    CanCreateAuthor = await AuthorizationService
        .IsGrantedAsync(BookStorePermissions.Authors.Create);

    CanEditAuthor = await AuthorizationService
        .IsGrantedAsync(BookStorePermissions.Authors.Edit);

    CanDeleteAuthor = await AuthorizationService
        .IsGrantedAsync(BookStorePermissions.Authors.Delete);
}

private async Task GetAuthorsAsync()
{
    var result = await AuthorAppService.GetListAsync(
        new GetAuthorListDto
        {
            MaxResultCount = PageSize,
            SkipCount = CurrentPage * PageSize,
            Sorting = CurrentSorting
        }
    );

    AuthorList = result.Items;
    TotalCount = (int)result.TotalCount;
}

private async Task
OnDataGridReadAsync(DataGridReadEventArgs<AuthorDto> e)
{
    CurrentSorting = e.Columns
        .Where(c => c.SortDirection != SortDirection.Default)
        .Select(c => c.Field + (c.SortDirection ==
SortDirection.Descending ? " DESC" : ""))
        .JoinAsString(",");
    CurrentPage = e.Page - 1;

    await GetAuthorsAsync();
}

```

```

        await InvokeAsync(StateHasChanged);
    }

private void OpenCreateAuthorModal()
{
    CreateValidationsRef.ClearAll();

    NewAuthor = new CreateAuthorDto();
    CreateAuthorModal.Show();
}

private void CloseCreateAuthorModal()
{
    CreateAuthorModal.Hide();
}

private void OpenEditAuthorModal(AuthorDto author)
{
    EditValidationsRef.ClearAll();

    EditingAuthorId = author.Id;
    EditingAuthor = ObjectMapper.Map<AuthorDto, UpdateAuthorDto>(author);
    EditAuthorModal.Show();
}

private async Task DeleteAuthorAsync(AuthorDto author)
{
    var confirmMessage = L["AuthorDeletionConfirmationMessage",
author.Name];
    if (!await Message.Confirm(confirmMessage))
    {
        return;
    }

    await AuthorAppService.DeleteAsync(author.Id);
    await GetAuthorsAsync();
}

private void CloseEditAuthorModal()
{
    EditAuthorModal.Hide();
}

private async Task CreateAuthorAsync()
{
    if (await CreateValidationsRef.ValidateAll())
    {
        await AuthorAppService.CreateAsync(NewAuthor);
        await GetAuthorsAsync();
        CreateAuthorModal.Hide();
    }
}

private async Task UpdateAuthorAsync()
{
    if (await EditValidationsRef.ValidateAll())
    {

```

```
        await AuthorAppService.UpdateAsync(EditAuthorId,
EditingAuthor);
        await GetAuthorsAsync();
        EditAuthorModal.Hide();
    }
}
}...
```

This class typically defines the properties and methods used by the `Authors.razor` page.

Object Mapping

`Authors` class uses the `IObjectMapper` in the `OpenEditAuthorModal` method. So, we need to define this mapping.

Open the `BookStoreBlazorAutoMapperProfile.cs` in the `Acme.BookStore.Blazor` project and add the following mapping code in the constructor:

```
```csharp
CreateMap<AuthorDto, UpdateAuthorDto>();
```
```

You will need to declare a `using Acme.BookStore.Authors;` statement to the beginning of the file.

Add to the Main Menu

Open the `BookStoreMenuContributor.cs` in the `Acme.BookStore.Blazor` project and add the following code to the end of the `ConfigureMainMenuAsync` method:

```
```csharp
if (await context.IsGrantedAsync(BookStorePermissions.Authors.Default))
{
 bookStoreMenu.AddItem(new ApplicationMenuItem(
 "BooksStore.Authors",
 l["Menu:Authors"],
 url: "/authors"
));
}
```
```

Localizations

We should complete the localizations we've used above. Open the `en.json` file under the `Localization/BookStore` folder of the `Acme.BookStore.Domain.Shared` project and add the following entries:

```
```json
"Menu:Authors": "Authors",
"Authors": "Authors",
"AuthorDeletionConfirmationMessage": "Are you sure to delete the author '{0}'?",
"BirthDate": "Birth date",
"NewAuthor": "New author"
```
```

Run the Application

Run and login to the application. **If you don't see the Authors menu item under the Book Store menu, that means you don't have the permission yet.** Go to the `'identity/roles'` page, click to the `*Actions*` button and select the `*Permissions*` action for the `**admin role**`:

 [bookstore-author-permissions](images/bookstore-author-permissions-2.png)

As you see, the admin role has no `*Author Management*` permissions yet. Click to the checkboxes and save the modal to grant the necessary permissions. You will see the `*Authors*` menu item under the `*Book Store*` in the main menu, after `**refreshing the page**`:

 [bookstore-authors-page](images/bookstore-authors-page-3.png)

That's all! This is a fully working CRUD page, you can create, edit and delete the authors.

> **Tip**: If you run the `'.DbMigrator'` console application after defining a new permission, it automatically grants these new permissions to the admin role and you don't need to manually grant the permissions yourself.

`{}{end}}`

The Next Part

See the [\[next part\]](#)(Part-10.md) of this tutorial.

3.1.10 10: Book to Author Relation

```
# Web Application Development Tutorial - Part 10: Book to Author Relation
```json
//[doc-params]
{
 "UI": ["MVC", "Blazor", "BlazorServer", "NG"],
 "DB": ["EF", "Mongo"]
}
```

```

About This Tutorial

In this tutorial series, you will build an ABP based web application named `'Acme.BookStore'`. This application is used to manage a list of books and their authors. It is developed using the following technologies:

- * `**{{DB_Value}}**` as the ORM provider.
- * `**{{UI_Value}}**` as the UI Framework.

This tutorial is organized as the following parts;

- [\[Part 1: Creating the server side\]](#)(Part-1.md)
- [\[Part 2: The book list page\]](#)(Part-2.md)
- [\[Part 3: Creating, updating and deleting books\]](#)(Part-3.md)
- [\[Part 4: Integration tests\]](#)(Part-4.md)
- [\[Part 5: Authorization\]](#)(Part-5.md)
- [\[Part 6: Authors: Domain layer\]](#)(Part-6.md)

- [Part 7: Authors: Database Integration](Part-7.md)
- [Part 8: Authors: Application Layer](Part-8.md)
- [Part 9: Authors: User Interface](Part-9.md)
- **Part 10: Book to Author Relation (this part)**

Download the Source Code

This tutorial has multiple versions based on your **UI** and **Database** preferences. We've prepared a few combinations of the source code to be downloaded:

- * [MVC (Razor Pages) UI with EF Core](https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore)
- * [Blazor UI with EF Core](https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore)
- * [Angular UI with MongoDB](https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb)

> If you encounter the "filename too long" or "unzip" error on Windows, please see [this guide](../KB/Windows-Path-Too-Long-Fix.md).

Introduction

We have created 'Book' and 'Author' functionalities for the book store application. However, currently there is no relation between these entities.

In this tutorial, we will establish a **1 to N** relation between the 'Author' and the 'Book' entities.

Add Relation to The Book Entity

Open the 'Books/Book.cs' in the 'Acme.BookStore.Domain' project and add the following property to the 'Book' entity:

```
```csharp
public Guid AuthorId { get; set; }

{{if DB=="EF"}}
> In this tutorial, we preferred to not add a **navigation property** to the 'Author' entity from the 'Book' class (like 'public Author Author { get; set; }'). This is due to follow the DDD best practices (rule: refer to other aggregates only by id). However, you can add such a navigation property and configure it for the EF Core. In this way, you don't need to write join queries while getting books with their authors (like we will be doing below) which makes your application code simpler.

{{end}}}
```

### ## Database & Data Migration

Added a new, required 'AuthorId' property to the 'Book' entity. But, \*\*what about the existing books\*\* on the database? They currently don't have 'AuthorId's and this will be a problem when we try to run the application.

This is a \*\*typical migration problem\*\* and the decision depends on your case;

- \* If you haven't published your application to the production yet, you can just delete existing books in the database, or you can even delete the entire database in your development environment.
- \* You can update the existing data programmatically on data migration or seed phase.
- \* You can manually handle it on the database.

We prefer to \*\*delete the database\*\* {{if DB=="EF"}}(you can run the `Drop-Database` in the \*Package Manager Console\*){{end}} since this is just an example project and data loss is not important. Since this topic is not related to the ABP Framework, we don't go deeper for all the scenarios.

```
{{if DB=="EF"}}
Update the EF Core Mapping
```

Locate to `OnModelCreating` method in the `BookStoreDbContext` class that under the `EntityFrameworkCore` folder of the `Acme.BookStore.EntityFrameworkCore` project and change the `builder.Entity<Book>` part as shown below:

```
```csharp
builder.Entity<Book>(b =>
{
    b.ToTable(BookStoreConsts.DbTablePrefix + "Books",
BookStoreConsts.DbSchema);
    b.ConfigureByConvention(); //auto configure for the base class props
    b.Property(x => x.Name).IsRequired().HasMaxLength(128);

    // ADD THE MAPPING FOR THE RELATION
    b.HasOne<Author>().WithMany().HasForeignKey(x =>
x.AuthorId).IsRequired();
});
```

Add New EF Core Migration

The startup solution is configured to use [Entity Framework Core Code First Migrations](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). Since we've changed the database mapping configuration, we should create a new migration and apply changes to the database.

Open a command-line terminal in the directory of the `Acme.BookStore.EntityFrameworkCore` project and type the following command:

```
```bash
dotnet ef migrations add Added_AuthorId_To_Book
```
```

This should create a new migration class with the following code in its `Up` method:

```
```csharp
migrationBuilder.AddColumn<Guid>(
 name: "AuthorId",
 table: "AppBooks",
 type: "uniqueidentifier",
```

```

 nullable: false,
 defaultValue: new Guid("00000000-0000-0000-0000-000000000000"));

migrationBuilder.CreateIndex(
 name: "IX_AppBooks_AuthorId",
 table: "AppBooks",
 column: "AuthorId");

migrationBuilder.AddForeignKey(
 name: "FK_AppBooks_AppAuthors_AuthorId",
 table: "AppBooks",
 column: "AuthorId",
 principalTable: "AppAuthors",
 principalColumn: "Id",
 onDelete: ReferentialAction.Cascade);
```
* Adds an `AuthorId` field to the `AppBooks` table.
* Creates an index on the `AuthorId` field.
* Declares the foreign key to the `AppAuthors` table.

> If you are using Visual Studio, you may want to use `Add-Migration Added_AuthorId_To_Book -c BookStoreDbContext` and `Update-Database -Context BookStoreDbContext` commands in the *Package Manager Console (PMC)*. In this case, ensure that {{if UI=="MVC"}}`Acme.BookStore.Web`{{else if UI=="BlazorServer"}}`Acme.BookStore.Blazor`{{else if UI=="Blazor" || UI=="NG"}}`Acme.BookStore.HttpApi.Host`{{end}} is the startup project and `Acme.BookStore.EntityFrameworkCore` is the *Default Project* in PMC.

```
{{end}}
Change the Data Seeder

```

Since the `AuthorId` is a required property of the `Book` entity, current data seeder code can not work. Open the `BookStoreDataSeederContributor` in the `Acme.BookStore.Domain` project and change as the following:

```

```csharp
using System;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Books;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore;

public class BookStoreDataSeederContributor
    : IDataSeedContributor, ITransientDependency
{
    private readonly IRepository<Book, Guid> _bookRepository;
    private readonly IAuthorRepository _authorRepository;
    private readonly AuthorManager _authorManager;

    public BookStoreDataSeederContributor(
        IRepository<Book, Guid> bookRepository,
        IAuthorRepository authorRepository,

```

```

        AuthorManager authorManager)
{
    _bookRepository = bookRepository;
    _authorRepository = authorRepository;
    _authorManager = authorManager;
}

public async Task SeedAsync(DataSeedContext context)
{
    if (await _bookRepository.GetCountAsync() > 0)
    {
        return;
    }

    var orwell = await _authorRepository.InsertAsync(
        await _authorManager.CreateAsync(
            "George Orwell",
            new DateTime(1903, 06, 25),
            "Orwell produced literary criticism and poetry, fiction and
polemical journalism; and is best known for the allegorical novella Animal
Farm (1945) and the dystopian novel Nineteen Eighty-Four (1949)."
        )
    );

    var douglas = await _authorRepository.InsertAsync(
        await _authorManager.CreateAsync(
            "Douglas Adams",
            new DateTime(1952, 03, 11),
            "Douglas Adams was an English author, screenwriter, essayist,
humorist, satirist and dramatist. Adams was an advocate for environmentalism
and conservation, a lover of fast cars, technological innovation and the
Apple Macintosh, and a self-proclaimed 'radical atheist'."
        )
    );

    await _bookRepository.InsertAsync(
        new Book
        {
            AuthorId = orwell.Id, // SET THE AUTHOR
            Name = "1984",
            Type = BookType.Dystopia,
            PublishDate = new DateTime(1949, 6, 8),
            Price = 19.84f
        },
        autoSave: true
    );

    await _bookRepository.InsertAsync(
        new Book
        {
            AuthorId = douglas.Id, // SET THE AUTHOR
            Name = "The Hitchhiker's Guide to the Galaxy",
            Type = BookType.ScienceFiction,
            PublishDate = new DateTime(1995, 9, 27),
            Price = 42.0f
        },
        autoSave: true
    );
}

```

```
        }
    }...
```

The only change is that we set the `AuthorId` properties of the `Book` entities.

> Delete existing books or delete the database before executing the `DbMigrator`. See the **Database & Data Migration** section above for more info.

```
{{if DB=="EF"}}
```

You can now run the `DbMigrator` console application to **migrate** the **database schema** and **seed** the initial data.

```
{{else if DB=="Mongo"}}
```

You can now run the `DbMigrator` console application to **seed** the initial data.

```
{{end}}
```

Application Layer

We will change the `BookAppService` to support the Author relation.

Data Transfer Objects

Let's begin from the DTOs.

BookDto

Open the `BookDto` class in the `Books` folder of the `Acme.BookStore.Application.Contracts` project and add the following properties:

```
```csharp
public Guid AuthorId { get; set; }
public string AuthorName { get; set; }
````
```

The final `BookDto` class should be following:

```
```csharp
using System;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Books;

public class BookDto : AuditedEntityDto<Guid>
{
 public Guid AuthorId { get; set; }

 public string AuthorName { get; set; }

 public string Name { get; set; }
}
```

```
 public BookType Type { get; set; }

 public DateTime PublishDate { get; set; }

 public float Price { get; set; }
}..
```

#### #### CreateUpdateBookDto

Open the `CreateUpdateBookDto` class in the `Books` folder of the `Acme.BookStore.Application.Contracts` project and add an `AuthorId` property as shown:

```
```csharp
public Guid AuthorId { get; set; }
```
```

#### #### AuthorLookupDto

Create a new class, `AuthorLookupDto`, inside the `Books` folder of the `Acme.BookStore.Application.Contracts` project:

```
```csharp
using System;
using Volo.Abp.Application.Dtos;

namespace Acme.BookStore.Books;

public class AuthorLookupDto : EntityDto<Guid>
{
    public string Name { get; set; }
}..
```

This will be used in a new method that will be added to the `IBookAppService`.

IBookAppService

Open the `IBookAppService` interface in the `Books` folder of the `Acme.BookStore.Application.Contracts` project and add a new method, named `GetAuthorLookupAsync`, as shown below:

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;

namespace Acme.BookStore.Books;

public interface IBookAppService :
 ICrudAppService< //Defines CRUD methods
 BookDto, //Used to show books
 Guid, //Primary key of the book entity
 PagedAndSortedResultRequestDto, //Used for paging/sorting
 CreateUpdateBookDto> //Used to create/update a book
{..}
```

```

{
 // ADD the NEW METHOD
 Task<ListResultDto<AuthorLookupDto>> GetAuthorLookupAsync();
}
...

```

This new method will be used from the UI to get a list of authors and fill a dropdown list to select the author of a book.

### ### BookAppService

Open the `BookAppService` class in the `Books` folder of the `Acme.BookStore.Application` project and replace the file content with the following code:

```

{{if DB=="EF"}}
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Linq.Dynamic.Core;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Permissions;
using Microsoft.AspNetCore.Authorization;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Entities;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Books;

[Authorize(BookStorePermissions.Books.Default)]
public class BookAppService :
    CrudAppService<
        Book, //The Book entity
        BookDto, //Used to show books
        Guid, //Primary key of the book entity
        PagedAndSortedResultRequestDto, //Used for paging/sorting
        CreateUpdateBookDto>, //Used to create/update a book
        IBookAppService //implement the IBookAppService
{
    private readonly IAuthorRepository _authorRepository;

    public BookAppService(
        IRepository<Book, Guid> repository,
        IAuthorRepository authorRepository)
        : base(repository)
    {
        _authorRepository = authorRepository;
        GetPolicyName = BookStorePermissions.Books.Default;
        GetListPolicyName = BookStorePermissions.Books.Default;
        CreatePolicyName = BookStorePermissions.Books.Create;
        UpdatePolicyName = BookStorePermissions.Books.Edit;
        DeletePolicyName = BookStorePermissions.Books.Delete;
    }
}
```

```

```

public override async Task<BookDto> GetAsync(Guid id)
{
 //Get the IQueryable<Book> from the repository
 var queryable = await Repository.GetQueryableAsync();

 //Prepare a query to join books and authors
 var query = from book in queryable
 join author in await _authorRepository.GetQueryableAsync() on
book.AuthorId equals author.Id
 where book.Id == id
 select new { book, author };

 //Execute the query and get the book with author
 var queryResult = await AsyncExecuter.FirstOrDefaultAsync(query);
 if (queryResult == null)
 {
 throw new EntityNotFoundException(typeof(Book), id);
 }

 var bookDto = ObjectMapper.Map<Book, BookDto>(queryResult.book);
 bookDto.AuthorName = queryResult.author.Name;
 return bookDto;
}

public override async Task<PagedResultDto<BookDto>>
GetListAsync(PagedAndSortedResultRequestDto input)
{
 //Get the IQueryable<Book> from the repository
 var queryable = await Repository.GetQueryableAsync();

 //Prepare a query to join books and authors
 var query = from book in queryable
 join author in await _authorRepository.GetQueryableAsync() on
book.AuthorId equals author.Id
 select new {book, author};

 //Paging
 query = query
 .OrderBy(NormalizeSorting(input.Sorting))
 .Skip(input.SkipCount)
 .Take(input.MaxResultCount);

 //Execute the query and get a list
 var queryResult = await AsyncExecuter.ToListAsync(query);

 //Convert the query result to a list of BookDto objects
 var bookDtos = queryResult.Select(x =>
 {
 var bookDto = ObjectMapper.Map<Book, BookDto>(x.book);
 bookDto.AuthorName = x.author.Name;
 return bookDto;
 }).ToList();

 //Get the total count with another query
 var totalCount = await Repository.GetCountAsync();

 return new PagedResultDto<BookDto>(
 totalCount,

```

```

 bookDtos
);
}

public async Task<ListResultDto<AuthorLookupDto>> GetAuthorLookupAsync()
{
 var authors = await _authorRepository.GetListAsync();

 return new ListResultDto<AuthorLookupDto>(
 ObjectMapper.Map<List<Author>, List<AuthorLookupDto>>(authors)
);
}

private static string NormalizeSorting(string sorting)
{
 if (sorting.IsNullOrEmpty())
 {
 return $"book.{nameof(Book.Name)}";
 }

 if (sorting.Contains("authorName",
StringComparison.OrdinalIgnoreCase))
 {
 return sorting.Replace(
 "authorName",
 "author.Name",
 StringComparison.OrdinalIgnoreCase
);
 }

 return $"book.{sorting}";
}
...

```

Let's see the changes we've done:

- \* Added `'[Authorize(BookStorePermissions.Books.Default)]'` to authorize the methods we've newly added/overrode (remember, authorize attribute is valid for all the methods of the class when it is declared for a class).
- \* Injected `IAuthorRepository` to query from the authors.
- \* Overrode the `GetAsync` method of the base `CrudAppService`, which returns a single `BookDto` object with the given `id`.
  - \* Used a simple LINQ expression to join books and authors and query them together for the given book id.
  - \* Used `AsyncExecuter.FirstOrDefaultAsync(...)` to execute the query and get a result. It is a way to use asynchronous LINQ extensions without depending on the database provider API. Check the [repository documentation](../../Repositories.md) to understand why we've used it.
  - \* Throws an `EntityNotFoundException` which results an `HTTP 404` (not found) result if requested book was not present in the database.
  - \* Finally, created a `BookDto` object using the `ObjectMapper`, then assigning the `AuthorName` manually.
- \* Overrode the `GetListAsync` method of the base `CrudAppService`, which returns a list of books. The logic is similar to the previous method, so you can easily understand the code.

\* Created a new method: `GetAuthorLookupAsync`. This simple gets all the authors. The UI uses this method to fill a dropdown list and select an author while creating/editing books.

```

{{else if DB=="Mongo"}}
```
using System;
using System.Collections.Generic;
using System.Linq.Dynamic.Core;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Acme.BookStore.Permissions;
using Microsoft.AspNetCore.Authorization;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore.Books;

[Authorize(BookStorePermissions.Books.Default)]
public class BookAppService :
    CrudAppService<
        Book, //The Book entity
        BookDto, //Used to show books
        Guid, //Primary key of the book entity
        PagedAndSortedResultDto<Book>, //Used for paging/sorting
        CreateUpdateBookDto>, //Used to create/update a book
        IBookAppService //implement the IBookAppService
{
    private readonly IAuthorRepository _authorRepository;

    public BookAppService(
        IRepository<Book, Guid> repository,
        IAuthorRepository authorRepository)
        : base(repository)
    {
        _authorRepository = authorRepository;
        GetPolicyName = BookStorePermissions.Books.Default;
        GetListPolicyName = BookStorePermissions.Books.Default;
        CreatePolicyName = BookStorePermissions.Books.Create;
        UpdatePolicyName = BookStorePermissions.Books.Edit;
        DeletePolicyName = BookStorePermissions.Books.Create;
    }

    public async override Task<BookDto> GetAsync(Guid id)
    {
        var book = await Repository.GetAsync(id);
        var bookDto = ObjectMapper.Map<Book, BookDto>(book);

        var author = await _authorRepository.GetAsync(book.AuthorId);
        bookDto.AuthorName = author.Name;

        return bookDto;
    }

    public async override Task<PagedResultDto<BookDto>>

```

```

        GetListAsync(PagedAndSortedResultRequestDto input)
{
    //Set a default sorting, if not provided
    if (input.Sorting.IsNullOrWhiteSpace())
    {
        input.Sorting = nameof(Book.Name);
    }

    //Get the IQueryable<Book> from the repository
    var queryable = await Repository.GetQueryableAsync();

    //Get the books
    var books = await AsyncExecuter.ToListAsync(
        queryable
            .OrderBy(input.Sorting)
            .Skip(input.SkipCount)
            .Take(input.MaxResultCount)
    );

    //Convert to DTOs
    var bookDtos = ObjectMapper.Map<List<Book>, List<BookDto>>(books);

    //Get a lookup dictionary for the related authors
    var authorDictionary = await GetAuthorDictionaryAsync(books);

    //Set AuthorName for the DTOs
    bookDtos.ForEach(bookDto => bookDto.AuthorName =
        authorDictionary[bookDto.AuthorId].Name);

    //Get the total count with another query (required for the paging)
    var totalCount = await Repository.GetCountAsync();

    return new PagedList<BookDto>(
        totalCount,
        bookDtos
    );
}

public async Task<ListResultDto<AuthorLookupDto>> GetAuthorLookupAsync()
{
    var authors = await _authorRepository.GetListAsync();

    return new ListResultDto<AuthorLookupDto>(
        ObjectMapper.Map<List<Author>, List<AuthorLookupDto>>(authors)
    );
}

private async Task<Dictionary<Guid, Author>>
    GetAuthorDictionaryAsync(List<Book> books)
{
    var authorIds = books
        .Select(b => b.AuthorId)
        .Distinct()
        .ToArray();

    var queryable = await _authorRepository.GetQueryableAsync();

    var authors = await AsyncExecuter.ToListAsync(

```

```

        queryable.Where(a => authorIds.Contains(a.Id))
    );
    return authors.ToDictionary(x => x.Id, x => x);
}
}..

```

Let's see the changes we've done:

- * Added `'[Authorize(BookStorePermissions.Books.Default)]'` to authorize the methods we've newly added/overrode (remember, authorize attribute is valid for all the methods of the class when it is declared for a class).
- * Injected `IAuthorRepository` to query from the authors.
- * Overrode the `GetAsync` method of the base `CrudAppService`, which returns a single `BookDto` object with the given `id`.
- * Overrode the `GetListAsync` method of the base `CrudAppService`, which returns a list of books. This code separately queries the authors from database and sets the name of the authors in the application code. Instead, you could create a custom repository method and perform a join query or take the power of the MongoDB API to get the books and their authors in a single query, which would be more performant.
- * Created a new method: `GetAuthorLookupAsync`. This simple gets all the authors. The UI uses this method to fill a dropdown list and select and author while creating/editing books.

```
{{end}}
```

Object to Object Mapping Configuration

Introduced the `AuthorLookupDto` class and used object mapping inside the `GetAuthorLookupAsync` method. So, we need to add a new mapping definition inside the `BookStoreApplicationAutoMapperProfile.cs` file of the `Acme.BookStore.Application` project:

```
```csharp
CreateMap<Author, AuthorLookupDto>();
```

```

Unit Tests

Some of the unit tests will fail since we made some changes on the `AuthorAppService`. Open the `BookAppService_Tests` in the `Books` folder of the `Acme.BookStore.Application.Tests` project and change the content as the following:

```
```csharp
using System;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Authors;
using Shouldly;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Validation;
using Xunit;

namespace Acme.BookStore.Books;
```

```

{{if DB=="Mongo"}}
[Collection(BookStoreTestConsts.CollectionDefinitionName)]{{end}}
public class BookAppService_Tests : BookStoreApplicationTestBase
{
 private readonly IBookAppService _bookAppService;
 private readonly IAuthorAppService _authorAppService;

 public BookAppService_Tests()
 {
 _bookAppService = GetRequiredService<IBookAppService>();
 _authorAppService = GetRequiredService<IAuthorAppService>();
 }

 [Fact]
 public async Task Should_Get_List_Of_Books()
 {
 //Act
 var result = await _bookAppService.GetListAsync(
 new PagedAndSortedResultRequestDto()
);

 //Assert
 result.TotalCount.ShouldBeGreaterThan(0);
 result.Items.ShouldContain(b => b.Name == "1984" &&
 b.AuthorName == "George Orwell");
 }

 [Fact]
 public async Task Should_Create_A_Valid_Book()
 {
 var authors = await _authorAppService.GetListAsync(new
GetAuthorListDto());
 var firstAuthor = authors.Items.First();

 //Act
 var result = await _bookAppService.CreateAsync(
 new CreateUpdateBookDto
 {
 AuthorId = firstAuthor.Id,
 Name = "New test book 42",
 Price = 10,
 PublishDate = System.DateTime.Now,
 Type = BookType.ScienceFiction
 }
);

 //Assert
 result.Id.ShouldNotBe(Guid.Empty);
 result.Name.ShouldBe("New test book 42");
 }

 [Fact]
 public async Task Should_Not_Create_A_Book_Without_Name()
 {
 var exception = await
Assert.ThrowsAsync<AbpValidationException>(async () =>
{
 await _bookAppService.CreateAsync(

```

```

 new CreateUpdateBookDto
 {
 Name = "",
 Price = 10,
 PublishDate = DateTime.Now,
 Type = BookType.ScienceFiction
 }
);
});

exception.ValidationErrors
 .ShouldContain(err => err.MemberNames.Any(m => m == "Name"));
}
```
* Changed the assertion condition in the `Should_Get_List_Of_Books` from `b => b.Name == "1984"` to `b => b.Name == "1984" && b.AuthorName == "George Orwell"` to check if the author name was filled.
* Changed the `Should_Create_A_Valid_Book` method to set the `AuthorId` while creating a new book, since it is required anymore.

```

The User Interface

```
{{if UI=="MVC"}}
```

The Book List

Book list page change is trivial. Open the `Pages/Books/Index.js` in the `Acme.BookStore.Web` project and add an `authorName` column between the `name` and `type` columns:

```
```js
...
{
 title: l('Name'),
 data: "name"
},
// ADDED the NEW AUTHOR NAME COLUMN
{
 title: l('Author'),
 data: "authorName"
},
{
 title: l('Type'),
 data: "type",
 render: function (data) {
 return l('Enum:BookType.' + data);
 }
},
```

```

When you run the application, you can see the **Author** column on the table:

![bookstore-added-author-to-book-list](images/bookstore-added-author-to-book-list-2.png)

Create Modal

Open the `Pages/Books/CreateModal.cshtml.cs` in the `Acme.BookStore.Web` project and change the file content as shown below:

```
```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Form;

namespace Acme.BookStore.Web.Pages.Books;

public class CreateModalModel : BookStorePageModel
{
 [BindProperty]
 public CreateBookViewModel Book { get; set; }

 public List<SelectListItem> Authors { get; set; }

 private readonly IBookAppService _bookAppService;

 public CreateModalModel(
 IBookAppService bookAppService)
 {
 _bookAppService = bookAppService;
 }

 public async Task OnGetAsync()
 {
 Book = new CreateBookViewModel();

 var authorLookup = await _bookAppService.GetAuthorLookupAsync();
 Authors = authorLookup.Items
 .Select(x => new SelectListItem(x.Name, x.Id.ToString()))
 .ToList();
 }

 public async Task<IActionResult> OnPostAsync()
 {
 await _bookAppService.CreateAsync(
 ObjectMapper.Map<CreateBookViewModel, CreateUpdateBookDto>(Book));
 return NoContent();
 }

 public class CreateBookViewModel
 {
 [SelectItems(nameof(Authors))]
 }
}
```

```

 [DisplayName("Author")]
 public Guid AuthorId { get; set; }

 [Required]
 [StringLength(128)]
 public string Name { get; set; }

 [Required]
 public BookType Type { get; set; } = BookType.Undefined;

 [Required]
 [DataType(DataType.Date)]
 public DateTime PublishDate { get; set; } = DateTime.Now;

 [Required]
 public float Price { get; set; }
}
```
* Changed type of the `Book` property from `CreateUpdateBookDto` to the new `CreateBookViewModel` class defined in this file. The main motivation of this change to customize the model class based on the User Interface (UI) requirements. We didn't want to use UI-related `[SelectItems(nameof(Authors))` and `[DisplayName("Author")]` attributes inside the `CreateUpdateBookDto` class.
* Added `Authors` property that is filled inside the `OnGetAsync` method using the `IBookAppService.GetAuthorLookupAsync` method defined before.
* Changed the `OnPostAsync` method to map `CreateBookViewModel` object to a `CreateUpdateBookDto` object since `IBookAppService.CreateAsync` expects a parameter of this type.

```

Edit Modal

Open the `Pages/Books/EditModal.cshtml.cs` in the `Acme.BookStore.Web` project and change the file content as shown below:

```

```csharp
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;
using Acme.BookStore.Books;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Form;

namespace Acme.BookStore.Web.Pages.Books;

public class EditModalModel : BookStorePageModel
{
 [BindProperty]
 public EditBookViewModel Book { get; set; }

 public List<SelectListItem> Authors { get; set; }
}

```

```

private readonly IBookAppService _bookAppService;

public EditModalModel(IBookAppService bookAppService)
{
 _bookAppService = bookAppService;
}

public async Task OnGetAsync(Guid id)
{
 var bookDto = await _bookAppService.GetAsync(id);
 Book = ObjectMapper.Map<BookDto, EditBookViewModel>(bookDto);

 var authorLookup = await _bookAppService.GetAuthorLookupAsync();
 Authors = authorLookup.Items
 .Select(x => new SelectListItem(x.Name, x.Id.ToString()))
 .ToList();
}

public async Task<IActionResult> OnPostAsync()
{
 await _bookAppService.UpdateAsync(
 Book.Id,
 ObjectMapper.Map<EditBookViewModel, CreateUpdateBookDto>(Book)
);

 return NoContent();
}

public class EditBookViewModel
{
 [HiddenInput]
 public Guid Id { get; set; }

 [SelectItems(nameof(Authors))]
 [DisplayName("Author")]
 public Guid AuthorId { get; set; }

 [Required]
 [StringLength(128)]
 public string Name { get; set; }

 [Required]
 public BookType Type { get; set; } = BookType.Undefined;

 [Required]
 [DataType(DataType.Date)]
 public DateTime PublishDate { get; set; } = DateTime.Now;

 [Required]
 public float Price { get; set; }
}
...
}

```

\* Changed type of the `Book` property from `CreateUpdateBookDto` to the new `EditBookViewModel` class defined in this file, just like done before for the create modal above.

\* Moved the `Id` property inside the new `EditBookViewModel` class.

```
* Added 'Authors' property that is filled inside the 'OnGetAsync' method
using the 'IBookAppService.GetAuthorLookupAsync' method.
* Changed the 'OnPostAsync' method to map 'EditBookViewModel' object to a
'CreateUpdateBookDto' object since 'IBookAppService.UpdateAsync' expects a
parameter of this type.
```

These changes require a small change in the 'EditModal.cshtml'. Remove the '`<abp-input asp-for="Id" />`' tag since we no longer need to it (since moved it to the 'EditBookViewModel'). The final content of the 'EditModal.cshtml' should be following:

```
```html
@page
@using Acme.BookStore.Localization
@using Acme.BookStore.Web.Pages.Books
@using Microsoft.Extensions.Localization
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Modal
@model EditModalModel
@inject IStringLocalizer<BookStoreResource> L
 @{
     Layout = null;
}
<abp-dynamic-form abp-model="Book" asp-page="/Books/EditModal">
    <abp-modal>
        <abp-modal-header title="@L["Update"].Value"></abp-modal-header>
        <abp-modal-body>
            <abp-form-content />
        </abp-modal-body>
        <abp-modal-footer
buttons="@((AbpModalButtons.Cancel|AbpModalButtons.Save))"></abp-modal-footer>
    </abp-modal>
</abp-dynamic-form>
```

```

### ### Object to Object Mapping Configuration

The changes above requires to define some object to object mappings. Open the 'BookStoreWebAutoMapperProfile.cs' in the 'Acme.BookStore.Web' project and add the following mapping definitions inside the constructor:

```
```csharp
CreateMap<Pages.Books.CreateModalModel.CreateBookViewModel,
CreateUpdateBookDto>();
CreateMap<BookDto, Pages.Books.EditModalModel.EditBookViewModel>();
CreateMap<Pages.Books.EditModalModel.EditBookViewModel,
CreateUpdateBookDto>();
```

```

You can run the application and try to create a new book or update an existing book. You will see a drop down list on the create/update form to select the author of the book:

`![bookstore-added-authors-to-modals](images/bookstore-added-authors-to-modals-2.png)`

`{{else if UI=="NG"}}`

### ### Service Proxy Generation

Since the HTTP APIs have been changed, you need to update Angular client side [service proxies](../UI/Angular/Service-Proxies.md). Before running 'generate-proxy' command, your host must be up and running.

Run the following command in the 'angular' folder (you may need to stop the angular application):

```
```bash
abp generate-proxy -t ng
````
```

This command will update the service proxy files under the '/src/app/proxy/' folder.

### ### The Book List

Book list page change is trivial. Open the '/src/app/book/book.component.html' and add the following column definition between the 'Name' and 'Type' columns:

```
```html
<ngx-datatable-column
  [name]="'::Author' | abpLocalization"
  prop="authorName"
  [sortable]="false"
></ngx-datatable-column>
````
```

When you run the application, you can see the \*Author\* column on the table:

![bookstore-books-with-authorname-angular](images/bookstore-books-with-authorname-angular-2.png)

### ### Create/Edit Forms

The next step is to add an Author selection (dropdown) to the create/edit forms. The final UI will look like the one shown below:

![bookstore-angular-author-selection](images/bookstore-angular-author-selection-2.png)

Added the Author dropdown as the first element in the form.

Open the '/src/app/book/book.component.ts' and change the content as shown below:

```
```js
import { ListService, PagedResultDto } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';
import { BookService, BookDto, bookTypeOptions, AuthorLookupDto } from
  '@proxy/books';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';
import { NgbDateNativeAdapter, NgbDateAdapter } from '@ng-bootstrap/ng-
bootstrap';
import { ConfirmationService, Confirmation } from '@abp/ng.theme.shared';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
````
```

```

@Component({
 selector: 'app-book',
 templateUrl: './book.component.html',
 styleUrls: ['./book.component.scss'],
 providers: [ListService, { provide: NgbDateAdapter, useClass: NgbDateNativeAdapter }],
})
export class BookComponent implements OnInit {
 book = { items: [], totalCount: 0 } as PagedResultDto<BookDto>;
 form: FormGroup;
 selectedBook = {} as BookDto;
 authors$: Observable<AuthorLookupDto[]>;
 bookTypes = bookTypeOptions;
 isModalOpen = false;

 constructor(
 public readonly list: ListService,
 private bookService: BookService,
 private fb: FormBuilder,
 private confirmation: ConfirmationService
) {
 this.authors$ = bookService.getAuthorLookup().pipe(map((r) => r.items));
 }

 ngOnInit() {
 const bookStreamCreator = (query) => this.bookService.getList(query);

 this.list.hookToQuery(bookStreamCreator).subscribe((response) => {
 this.book = response;
 });
 }

 createBook() {
 this.selectedBook = {} as BookDto;
 this.buildForm();
 this.isModalOpen = true;
 }

 editBook(id: string) {
 this.bookService.get(id).subscribe((book) => {
 this.selectedBook = book;
 this.buildForm();
 this.isModalOpen = true;
 });
 }

 buildForm() {
 this.form = this.fb.group({
 authorId: [this.selectedBook.authorId || null, Validators.required],
 name: [this.selectedBook.name || null, Validators.required],
 type: [this.selectedBook.type || null, Validators.required],
 publishDate: [

```

```

 this.selectedBook.publishDate ? new
 Date(this.selectedBook.publishDate) : null,
 Validators.required,
],
 price: [this.selectedBook.price || null, Validators.required],
);
}

save() {
 if (this.form.invalid) {
 return;
 }

 const request = this.selectedBook.id
 ? this.bookService.update(this.selectedBook.id, this.form.value)
 : this.bookService.create(this.form.value);

 request.subscribe(() => {
 this.isModalOpen = false;
 this.form.reset();
 this.list.get();
 });
}

delete(id: string) {
 this.confirmation.warn('::AreYouSureToDelete',
'AbpAccount::AreYouSure').subscribe((status) => {
 if (status === Confirmation.Status.confirm) {
 this.bookService.delete(id).subscribe(() => this.list.get());
 }
 });
}
...
}

* Added imports for the `AuthorLookupDto`, `Observable` and `map`.
* Added `authors$: Observable<AuthorLookupDto[]>;` field after the
`selectedBook`.
* Added `this.authors$ = bookService.getAuthorLookup().pipe(map((r) =>
r.items));` into the constructor.
* Added `authorId: [this.selectedBook.authorId || null,
Validators.required],` into the `buildForm()` function.

```

Open the `/src/app/book/book.component.html` and add the following form group just before the book name form group:

```

```html
<div class="form-group">
    <label for="author-id">Author</label><span> * </span>
    <select class="form-control" id="author-id" formControlName="authorId">
        <option [ngValue]="">Select author</option>
        <option [ngValue]="author.id" *ngFor="let author of authors$ | async">
            {{author.name}}
        </option>
    </select>
</div>
```

```

That's all. Just run the application and try to create or edit an author.

```
 {{end}}}

{{if UI == "Blazor" || UI == "BlazorServer"}}

The Book List
```

It is very easy to show the *\*Author Name\** in the book list. Open the `'/Pages/Books.razor'` file in the `'Acme.BookStore.Blazor'` project and add the following `'DataGridColumn'` definition just after the `'Name'` (book name) column:

```
```xml
<DataGridColumn TItem="BookDto"
    Field="@nameof(BookDto.AuthorName)"
    Caption="@L["Author"]"></DataGridColumn>
````
```

When you run the application, you can see the *\*Author\** column on the table:

([blazor-bookstore-book-list-with-authors](images/blazor-bookstore-book-list-with-authors-2.png))

```
Create Book Modal
```

Add the following field to the `'@code'` section of the `'Books.razor'` file:

```
```csharp
IReadOnlyList<AuthorLookupDto> authorList = Array.Empty<AuthorLookupDto>();
````
```

Override the `'OnInitializedAsync'` method and adding the following code:

```
```csharp
protected override async Task OnInitializedAsync()
{
    await base.OnInitializedAsync();
    authorList = (await AppService.GetAuthorLookupAsync()).Items;
}
````
```

\* It is essential to call the `'base.OnInitializedAsync()'` since `'AbpCrudPageBase'` has some initialization code to be executed.

Override the `'OpenCreateModalAsync'` method and adding the following code:

```
```csharp
protected override async Task OpenCreateModalAsync()
{
    if (!authorList.Any())
    {
        throw new UserFriendlyException(message:
L["AnAuthorIsRequiredForCreatingBook"]);
    }

    await base.OpenCreateModalAsync();
    NewEntity.AuthorId = authorList.First().Id;
}
```

```
} ...
```

The final `@code` block should be the following:

```
```csharp
@code
{
 //ADDED A NEW FIELD
 IReadOnlyList<AuthorLookupDto> authorList =
 Array.Empty<AuthorLookupDto>();

 public Books() // Constructor
 {
 CreatePolicyName = BookStorePermissions.Books.Create;
 UpdatePolicyName = BookStorePermissions.Books.Edit;
 DeletePolicyName = BookStorePermissions.Books.Delete;
 }

 //GET AUTHORS ON INITIALIZATION
 protected override async Task OnInitializedAsync()
 {
 await base.OnInitializedAsync();
 authorList = (await AppService.GetAuthorLookupAsync()).Items;
 }

 protected override async Task OpenCreateModalAsync()
 {
 if (!authorList.Any())
 {
 throw new UserFriendlyException(message:
L["AnAuthorIsRequiredForCreatingBook"]);
 }

 await base.OpenCreateModalAsync();
 NewEntity.AuthorId = authorList.First().Id;
 }
}
...```

```

Finally, add the following `Field` definition into the `ModalBody` of the *\*Create\** modal, as the first item, before the `Name` field:

```
```xml
<Field>
    <FieldLabel>@L["Author"]</FieldLabel>
    <Select TValue="Guid" @bind-SelectedValue="@NewEntity.AuthorId">
        @foreach (var author in authorList)
        {
            <SelectItem TValue="Guid" Value="@author.Id">
                @author.Name
            </SelectItem>
        }
    </Select>
</Field>
...```

```

This requires to add a new localization key to the `en.json` file:

```
```js
"AnAuthorIsRequiredForCreatingBook": "An author is required to create a book"
```
```

You can run the application to see the **Author Selection** while creating a new book:

![book-create-modal-with-author](images/book-create-modal-with-author-2.png)

Edit Book Modal

Add the following `'Field'` definition into the `'ModalBody'` of the **Edit** modal, as the first item, before the `'Name'` field:

```
```xml
<Field>
 <FieldLabel>@L["Author"]</FieldLabel>
 <Select TValue="Guid" @bind-SelectedValue="@EditingEntity.AuthorId">
 @foreach (var author in authorList)
 {
 <SelectItem TValue="Guid" Value="@author.Id">
 @author.Name
 </SelectItem>
 }
 </Select>
</Field>
```
```

That's all. We are reusing the `'authorList'` defined for the **Create** modal.

```
{{end}}
```

3.2 Community Articles

<https://community.abp.io/articles>

3.3 Migrating from the ASP.NET Boilerplate

```
# Migrating from ASP.NET Boilerplate to the ABP Framework
```

ABP Framework is ****the successor**** of the open source [[ASP.NET Boilerplate](#)](<https://aspnetboilerplate.com/>) framework. This guide aims to help you to ****migrate your existing solutions**** (you developed with the ASP.NET Boilerplate framework) to the ABP Framework.

Introduction

****ASP.NET Boilerplate**** is being ****actively developed**** [[since 2013](#)](<https://github.com/aspnetboilerplate/aspnetboilerplate/graphs/contributors>). It is loved, used and contributed by the community. It started as a side project of [[a developer](#)](<http://halilrahimkalkan.com/>), but now it is officially maintained and improved by the company [[Volosoft](#)](<https://volosoft.com/>) in addition to the great community support.

ABP Framework has the same goal of the ASP.NET Boilerplate framework: **Don't Repeat Yourself**! It provides infrastructure, tools and startup templates to make a developer's life easier while developing enterprise software solutions.

See [[the introduction blog post](#)](<https://blog.abp.io/abp-vNext-Announcement>) if you wonder why we needed to re-write the ASP.NET Boilerplate framework.

Should I Migrate?

No, you don't have to!

- * ASP.NET Boilerplate is still in active development and maintenance.
 - * It also works on the latest ASP.NET Core and related libraries and tools.
- It is up to date.

However, if you want to take the advantage of the new ABP Framework [[features](#)](<https://abp.io/features>) and the new architecture opportunities (like support for NoSQL databases, microservice compatibility, advanced modularity), you can use this document as a guide.

What About the ASP.NET Zero?

[[ASP.NET Zero](#)](<https://aspnetzero.com/>) is a commercial product developed by the core ASP.NET Boilerplate team, on top of the ASP.NET Boilerplate framework. It provides pre-built application [[features](#)](<https://aspnetzero.com/Features>), code generation tooling and a nice looking modern UI. It is trusted and used by thousands of companies from all around the World.

We have created the [[ABP Commercial](#)](<https://commercial.abp.io/>) as an alternative to the ASP.NET Zero. ABP Commercial is more modular and upgradeable compared to the ASP.NET Zero. It currently has less features compared to ASP.NET Zero, but the gap will be closed by the time (it also has some features don't exist in the ASP.NET Zero).

We think ASP.NET Zero is still a good choice while starting a new application. It is production ready and mature solution delivered as a full source code. It is being actively developed and we are constantly adding new features.

We don't suggest to migrate your ASP.NET Zero based solution to the ABP Commercial if;

- * Your ASP.NET Zero solution is mature and it is in maintenance rather than a rapid development.
- * You don't have enough development time to perform the migration.
- * A monolithic solution fits in your business.
- * You've customized existing ASP.NET Zero features too much based on your requirements.

We also suggest you to compare the features of two products based on your needs.

If you have an ASP.NET Zero based solution and want to migrate to the ABP Commercial, this guide will also help you.

ASP.NET MVC 5.x Projects

The ABP Framework doesn't support ASP.NET MVC 5.x, it only works with ASP.NET Core. So, if you migrate your ASP.NET MVC 5.x based projects, you will also deal with the .NET Core migration.

The Migration Progress

We've designed the ABP Framework by **getting the best parts** of the ASP.NET Boilerplate framework, so it will be familiar to you if you've developed ASP.NET Boilerplate based applications.

In the ASP.NET Boilerplate, we have not worked much on the UI side, but used some free themes (we've used [[metronic theme](#)](<https://keenthemes.com/metronic/>) for ASP.NET Zero on the other side). In the ABP Framework, we worked a lot on the UI side (especially for the MVC / Razor Pages UI, because Angular already has a good modular system of its own). So, the **most challenging part** of the migration will be the **User Interface** of your solution.

ABP Framework is (and ASP.NET Boilerplate was) designed based on the [[Domain Driven Design](#)](<https://docs.abp.io/en/abp/latest/Domain-Driven-Design>) patterns & principles and the startup templates are layered based on the DDD layers. So, this guide respects to that layering model and explains the migration layer by layer.

Creating the Solution

First step of the migration is to create a new solution. We suggest you to create a fresh new project using [[the startup templates](#)](<https://abp.io/get-started>) (see [[this document](#)](<https://docs.abp.io/en/commercial/latest/getting-started>) for the ABP Commercial).

After creating the project and running the application, you can copy your code from your existing solution to the new solution step by step, layer by layer.

About Pre-Built Modules

The startup projects for the ABP Framework use the [[pre-built modules](#)](<https://docs.abp.io/en/abp/latest/Modules/Index>) (not all of them, but the essentials) and themes as NuGet/NPM packages. So, you don't see the source code of the modules/themes in your solution. This has an advantage that you can easily update these packages when a new version is released. However, you can not easily customize them as their source code is in your hands.

We suggest to continue to use these modules as package references, in this way you can get new features easily (see [[abp update command](#)](<https://docs.abp.io/en/abp/latest/CLI#update>)). In this case, you have a few options to customize or extend the functionality of the used modules;

* You can create your own entity and share the same database table with an entity in a used module. An example of this is the '[AppUser](#)' entity comes in the startup template.

- * You can [replace](<https://docs.abp.io/en/abp/latest/Dependency-Injection#replace-a-service>) a domain service, application service, controller, page model or other types of services with your own implementation. We suggest you to inherit from the existing implementation and override the method you need.
- * You can replace a `*.cshtml` view, page, view component, partial view... with your own one using the [Virtual File System](<https://docs.abp.io/en/abp/latest/Virtual-File-System>).
- * You can override javascript, css, image or any other type of static files using the [Virtual File System](<https://docs.abp.io/en/abp/latest/Virtual-File-System>).

More extend/customization options will be developed and documented by the time. However, if you need to fully change the module implementation, it is best to add the [source code](<https://github.com/abpframework/abp/tree/dev/modules>) of the related module into your own solution and remove the package dependencies.

The source code of the modules and the themes are [MIT](<https://opensource.org/licenses/MIT>) licensed, you can fully own and customize it without any limitation (for the ABP Commercial, you can download the source code of a [module](<https://commercial.abp.io/modules>)/[theme](<https://commercial.abp.io/themes>) if you have a [license](<https://commercial.abp.io/pricing>) type that includes the source code).

The Domain Layer

Most of your domain layer code will remain same, while you need to perform some minor changes in your domain objects.

Aggregate Roots & Entities

The ABP Framework and the ASP.NET Boilerplate both have the `IEntity` and `IEntity<T>` interfaces and `Entity` and `Entity<T>` base classes to define entities but they have some differences.

If you have an entity in the ASP.NET Boilerplate application like that:

```
```csharp
public class Person : Entity //Default PK is int for the ASP.NET Boilerplate
{
 ...
}
```

Then your primary key (the `Id` property in the base class) is `int` which is the \*\*default primary key\*\* (PK) type for the ASP.NET Boilerplate. If you want to set another type of PK, you need to explicitly declare it:

```
```csharp
public class Person : Entity<Guid> //Set explicit PK in the ASP.NET
Boilerplate
{
    ...
}
```

ABP Framework behaves differently and expects to **always explicitly set** the PK type:

```
```csharp
public class Person : Entity<Guid> //Set explicit PK in the ASP.NET
Boilerplate
{
 ...
}
...```

```

'`Id`' property (and the corresponding PK in the database) will be '`Guid`' in this case.

#### #### Composite Primary Keys

ABP Framework also has a non-generic '`Entity`' base class, but this time it has no '`Id`' property. Its purpose is to allow you to create entities with composite PKs. See [[the documentation](#)](<https://docs.abp.io/en/abp/latest/Entities#entities-with-composite-keys>) to learn more about the composite PKs.

#### #### Aggregate Root

It is best practice now to use the '`AggregateRoot`' base class instead of '`Entity`' for aggregate root entities. See [[the documentation](#)](<https://docs.abp.io/en/abp/latest/Entities#aggregateroot-class>) to learn more about the aggregate roots.

In opposite to the ASP.NET Boilerplate, the ABP Framework creates default repositories ('`IRepository<T>`') \*\*only for the aggregate roots\*\*. It doesn't create for other types derived from the '`Entity`'.

If you still want to create default repositories for all entity types, find the `*YourProjectName*EntityFrameworkCoreModule` class in your solution and change '`options.AddDefaultRepositories()`' to '`options.AddDefaultRepositories(includeAllEntities: true)`' (it may be already like that for the application startup template).

#### #### Migrating the Existing Entities

We suggest & use the GUID as the PK type for all the ABP Framework modules. However, you can continue to use your existing PK types to migrate your database tables easier.

The challenging part will be the primary keys of the ASP.NET Boilerplate related entities, like Users, Roles, Tenants, Settings... etc. Our suggestion is to copy data from existing database to the new database tables using a tool or in a manual way (be careful about the foreign key values).

#### #### Documentation

See the documentation for details on the entities:

- \* [[ASP.NET Boilerplate - Entity documentation](#)](<https://aspnetboilerplate.com/Pages/Documents/Entities>)
- \* [[ABP Framework - Entity documentation](#)](<https://docs.abp.io/en/abp/latest/Entities>)

### ### Repositories

> ABP Framework creates default repositories (``IRepository<T>``) \*\*only for the aggregate roots\*\*. It doesn't create for other types derived from the `Entity`. See the "Aggregate Root" section above for more information.

The ABP Framework and the ASP.NET Boilerplate both have the default generic repository system, but has some differences.

### #### Injecting the Repositories

In the ASP.NET Boilerplate, there are two default repository interfaces you can directly inject and use:

- \* `IRepository< TEntity >` (e.g. `IRepository< Person >`) is used for entities with `int` primary key (PK) which is the default PK type.
- \* `IRepository< TEntity, TKey >` (e.g. `IRepository< Person, Guid >`) is used for entities with other types of PKs.

ABP Framework doesn't have a default PK type, so you need to \*\*explicitly declare the PK type\*\* of your entity, like `IRepository< Person, int >` or `IRepository< Person, Guid >`.

ABP Framework also has the `IRepository< TEntity >` (without PK), but it is mostly used when your entity has a composite PK (because this repository has no methods work with the `Id` property). See [[the documentation](#)](<https://docs.abp.io/en/abp/latest/Entities#entities-with-composite-keys>) to learn more about the \*\*composite PKs\*\*.

### #### Restricted Repositories

ABP Framework additionally provides a few repository interfaces:

- \* `IBasicRepository< TEntity, TKey >` has the same methods with the `IRepository` except it doesn't have `IQueryable` support. It can be useful if you don't want to expose complex querying code to the application layer. In this case, you typically want to create custom repositories to encapsulate the querying logic. It is also useful for database providers those don't support `IQueryable`.
- \* `IReadOnlyRepository< TEntity, TKey >` has the methods get data from the database, but doesn't contain any method change the database.
- \* `IReadOnlyBasicRepository< TEntity, TKey >` is similar to the read only repository but also doesn't support `IQueryable`.

All the interfaces also have versions without `TKey` (like `IReadOnlyRepository< TEntity >` ) those can be used for composite PKs just like explained above.

### #### GetAll() vs IQueryable

ASP.NET Boilerplate's repository has a `GetAll()` method that is used to obtain an `IQueryable` object to execute LINQ on it. An example application service calls the `GetAll()` method:

```
```csharp
public class PersonAppService : ApplicationService, IPersonAppService
{
```

```

private readonly IRepository<Person, Guid> _personRepository;

public PersonAppService(IRepository<Person, Guid> personRepository)
{
    _personRepository = personRepository;
}

public async Task DoIt()
{
    var people = await _personRepository
        .GetAll() //GetAll() returns IQueryable
        .Where(p => p.BirthYear > 2000) //Use LINQ extension methods
        .ToListAsync();
}
...

```

ABP Framework's repository have `GetQueryableAsync` instead:

```

```csharp
public class PersonAppService : ApplicationService, IPersonAppService
{
 private readonly IRepository<Person, Guid> _personRepository;

 public PersonAppService(IRepository<Person, Guid> personRepository)
 {
 _personRepository = personRepository;
 }

 public async Task DoIt()
 {
 var queryable = await _personRepository.GetQueryableAsync();
 var people = await queryable
 .Where(p => p.BirthYear > 2000) //Use LINQ extension methods
 .ToListAsync();
 }
}
...

```

> Note that in order to use the async LINQ extension methods (like `ToListAsync` here), you may need to depend on the database provider (like EF Core) since these methods are defined in the database provider package, they are not standard LINQ methods. See the [\[repository document\]](#)(Repositories.md) for alternative approaches for async query execution.

#### #### FirstOrDefault(predicate), Single()... Methods

ABP Framework repository has not such methods get predicate (expression) since the repository itself is `IQueryable` and all these methods are already standard LINQ extension methods those can be directly used.

However, it provides the following methods those can be used to query a single entity by its Id:

- \* `FindAsync(id)` returns the entity or null if not found.
- \* `GetAsync(id)` method returns the entity or throws an `EntityNotFoundException` (which causes HTTP 404 status code) if not found.

#### #### Sync vs Async

ABP Framework repository has no sync methods (like '`Insert`'). All the methods are async (like '`InsertAsync`'). So, if your application has sync repository method usages, convert them to async versions.

In general, ABP Framework forces you to completely use async everywhere, because mixing async & sync methods is not a recommended approach.

#### #### Documentation

See the documentation for details on the repositories:

- \* [ASP.NET Boilerplate – Repository documentation](<https://aspnetboilerplate.com/Pages/Documents/Repositories>)
- \* [ABP Framework – Repository documentation](<https://docs.abp.io/en/abp/latest/Repositories>)

#### ### Domain Services

Your domain service logic mostly remains same on the migration. ABP Framework also defines the base '`DomainService`' class and the '`IDomainService`' interface just works like the ASP.NET Boilerplate.

#### ## The Application Layer

Your application service logic remains similar on the migration. ABP Framework also defines the base '`ApplicationService`' class and the '`IApplicationService`' interface just works like the ASP.NET Boilerplate, but there are some differences in details.

#### ### Declarative Authorization

ASP.NET Boilerplate has '`AbpAuthorize`' and '`AbpMvcAuthorize`' attributes for declarative authorization. Example usage:

```
```csharp
[AbpAuthorize("MyUserDeletionPermissionName")]
public async Task DeleteUserAsync(...)
{
    ...
}
```

ABP Framework doesn't have such a custom attribute. It uses the standard '`Authorize`' attribute in all layers.

```
```csharp
[Authorize("MyUserDeletionPermissionName")]
public async Task DeleteUserAsync(...)
{
 ...
}
```

This is possible with the better integration to the Microsoft Authorization Extensions libraries. See the Authorization section below for more information about the authorization system.

### ### CrudAppService and AsyncCrudAppService Classes

ASP.NET Boilerplate has `CrudAppService` (with sync service methods) and `AsyncCrudAppService` (with async service methods) classes.

ABP Framework only has the `CrudAppService` which actually has only the async methods (instead of sync methods).

ABP Framework's `CrudAppService` method signatures are slightly different than the old one. For example, old update method signature was `Task< TEntityDto > UpdateAsync(TUpdateInput input)` while the new one is `Task< TGetOutputDto > UpdateAsync(TKey id, TUpdateInput input)`. The main difference is that it gets the Id of the updating entity as a separate parameter instead of including in the input DTO.

### ### Data Transfer Objects (DTOs)

There are similar base DTO classes (like `EntityDto`) in the ABP Framework too. So, you can find the corresponding DTO base class if you need.

### #### Validation

You can continue to use the data annotation attributes to validate your DTOs just like in the ASP.NET Boilerplate.

ABP Framework doesn't include the `ICustomValidate` that does exists in the ASP.NET Boilerplate. Instead, you should implement the standard `IValidatableObject` interface for your custom validation logic.

## ## The Infrastructure Layer

### ### Namespaces

ASP.NET Boilerplate uses the `Abp.\*` namespaces while the ABP Framework uses the `Volo.Abp.\*` namespaces for the framework and pre-built fundamental modules.

In addition, there are also some pre-built application modules (like docs and blog modules) those are using the `Volo.\*` namespaces (like `Volo.Blogging.\*` and `Volo.Docs.\*`). We consider these modules as standalone open source products developed by Volosoft rather than add-ons or generic modules completing the ABP Framework and used in the applications. We've developed them as a module to make them re-usable as a part of a bigger solution.

### ### Module System

Both of the ASP.NET Boilerplate and the ABP Framework have the `AbpModule` while they are a bit different.

ASP.NET Boilerplate's `AbpModule` class has `PreInitialize`, `Initialize` and `PostInitialize` methods you can override and configure the framework and the depended modules. You can also register and resolve dependencies in these methods.

ABP Framework's `AbpModule` class has the `ConfigureServices` and `OnApplicationInitialization` methods (and their Pre and Post versions). It is similar to ASP.NET Core's Startup class. You configure other services and

register dependencies in the `ConfigureServices`. However, you can now resolve dependencies in that point. You can resolve dependencies and configure the ASP.NET Core pipeline in the `OnApplicationInitialization` method while you can not register dependencies here. So, the new module classes separate dependency registration phase from dependency resolution phase since it follows the ASP.NET Core's approach.

### ### Dependency Injection

#### #### The DI Framework

ASP.NET Boilerplate is using the [Castle Windsor](<http://www.castleproject.org/projects/windsor/>) as the dependency injection framework. This is a fundamental dependency of the ASP.NET Boilerplate framework. We've got a lot of feedback to make the ASP.NET Boilerplate DI framework agnostic, but it was not so easy because of the design.

ABP Framework is dependency injection framework independent since it uses Microsoft's [Dependency Injection Extensions](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection>) library as an abstraction. None of the ABP Framework or module packages depends on any specific library.

However, ABP Framework doesn't use the Microsoft's base DI library because it has some missing features ABP Framework needs to: Property Injection and Interception. All the startup templates and the samples are using the [Autofac](<https://autofac.org/>) as the DI library and it is the only [officially integrated](Autofac-Integration.md) library to the ABP Framework. We suggest you to use the Autofac with the ABP Framework if you have not a good reason. If you have a good reason, please create an [issue](<https://github.com/abpframework/abp/issues/new>) on GitHub to request it or just implement it and send a pull request :)

#### #### Registering the Dependencies

Registering the dependencies are similar and mostly handled by the framework conventionally (like repositories, application services, controllers... etc). Implement the same `ITransientDependency`, `ISingletonDependency` and `IScopedDependency` interfaces for the services not registered by conventions.

When you need to manually register dependencies, use the `context.Services` in the `ConfigureServices` method of your module. Example:

```
```csharp
public class BlogModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        //Register an instance as singleton
        context.Services.AddSingleton<TaxCalculator>(new
        TaxCalculator(taxRatio: 0.18));

        //Register a factory method that resolves from IServiceProvider
        context.Services.AddScoped<ITaxCalculator>(
            sp => sp.GetService<TaxCalculator>()
        );
    }
}
```

```
    }
}
```

See the ABP Framework [dependency injection document](<https://docs.abp.io/en/abp/latest/Dependency-Injection>) for details.

Configuration vs Options System

ASP.NET Boilerplate has its own configuration system to configure the framework and the modules. For example, you could disable the audit logging in the `Initialize` method of your [module](<https://aspnetboilerplate.com/Pages/Documents/Module-System>):

```
```csharp
public override void Initialize()
{
 Configuration.Auditing.IsEnabled = false;
}
````
```

ABP Framework uses [the options pattern](Options.md) to configure the framework and the modules. You typically configure the options in the `ConfigureServices` method of your [module](Module-Development-Basics.md):

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 Configure<AbpAuditingOptions>(options =>
 {
 options.IsEnabled = false;
 });
}
````
```

Instead of a central configuration object, there are separated option classes for every module and feature those are defined in the related documents.

IAbpSession vs ICurrentUser and ICurrentTenant

ASP.NET Boilerplate's `IAbpSession` service is used to obtain the current user and tenant information, like `UserId` and `TenantId`.

ABP Framework doesn't have the same service. Instead, use `ICurrentUser` and `ICurrentTenant` services. These services are defined as base properties in some common classes (like `ApplicationService` and `AbpController`), so you generally don't need to manually inject them. They also have much properties compared to the `IAbpSession`.

Authorization

ABP Framework extends the [ASP.NET Core Authorization](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction>) by adding **permissions** as auto [policies](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies>) and allowing the

authorization system to be usable in the [application services](Application-Services.md) too.

AbpAuthorize vs Authorize

Use the standard `[Authorize]` and `[AllowAnonymous]` attributes instead of ASP.NET Boilerplate's custom `[AbpAuthorize]` and `[AbpAllowAnonymous]` attributes.

IPermissionChecker vs IAuthorizationService

Use the standard `IAuthorizationService` to check permissions instead of the ASP.NET Boilerplate's `IPermissionChecker` service. While `IPermissionChecker` also exists in the ABP Framework, it is used to explicitly use the permissions. Using `IAuthorizationService` is the recommended way since it covers other type of policy checks too.

AuthorizationProvider vs PermissionDefinitionProvider

You inherit from the `AuthorizationProvider` in the ASP.NET Boilerplate to define your permissions. ABP Framework replaces it by the `PermissionDefinitionProvider` base class. So, define your permissions by inheriting from the `PermissionDefinitionProvider` class.

Unit of Work

Unit of work system has been designed to work seamlessly. For most of the cases, you don't need to change anything.

`UnitOfWork` attribute of the ABP Framework doesn't have the `ScopeOption` (type of `TransactionScopeOption`) property. Instead, use `IUnitOfWorkManager.Begin()` method with `requiresNew = true` to create an independent inner transaction in a transaction scope.

Data Filters

ASP.NET Boilerplate implements the data filtering system as a part of the unit of work. ABP Framework has a separate `IDataFilter` service.

See the [data filtering document](Data-Filtering.md) to learn how to enable/disable a filter.

See [the UOW documentation](Unit-Of-Work.md) for more about the UOW system.

Multi-Tenancy

IMustHaveTenant & IMayHaveTenant vs IMultiTenant

ASP.NET Boilerplate defines `IMustHaveTenant` and `IMayHaveTenant` interfaces to implement them for your entities. In this way, your entities are automatically filtered according to the current tenant. Because of the design, there was a problem: You had to create a "Default" tenant in the database with "1" as the Id if you want to create a non multi-tenant application (this "Default" tenant was used as the single tenant).

ABP Framework has a single interface for multi-tenant entities: `IMultiTenant` which defines a nullable `TenantId` property of type `Guid`.

If your application is not multi-tenant, then your entities will have null TenantId (instead of a default one).

On the migration, you need to change the TenantId field type and replace these interfaces with the '[IMultiTenant](#)'

Switch Between Tenants

In some cases you might need to switch to a tenant for a code scope and work with the tenant's data in this scope.

In ASP.NET Boilerplate, it is done using the '[IUnitOfWorkManager](#)' service:

```
```csharp
public async Task<List<Product>> GetProducts(int tenantId)
{
 using (_unitOfWorkManager.Current.SetTenantId(tenantId))
 {
 return await _productRepository.GetAllListAsync();
 }
}
...```

```

In the ABP Framework it is done with the '[ICurrentTenant](#)' service:

```
```csharp
public async Task<List<Product>> GetProducts(Guid tenantId)
{
    using (_currentTenant.Change(tenantId))
    {
        return await _productRepository.GetListAsync();
    }
}
...```

```

Pass '`null`' to the '[Change](#)' method to switch to the host side.

Caching

ASP.NET Boilerplate has its [own distributed caching abstraction](<https://aspnetboilerplate.com/Pages/Documents/Caching>) which has in-memory and Redis implementations. You typically inject the '[ICacheManager](#)' service and use its '[GetCache\(...\)](#)' method to obtain a cache, then get and set objects in the cache.

ABP Framework uses and extends ASP.NET Core's [distributed caching abstraction]([Caching.md](#)). It defines the '[IDistributedCache<T>](#)' services to inject a cache and get/set objects.

Logging

ASP.NET Boilerplate uses Castle Windsor's [logging facility](<https://github.com/castleproject/Windsor/blob/master/docs/logging-facility.md>) as an abstraction and supports multiple logging providers including Log4Net (the default one comes with the startup projects) and Serilog. You typically property-inject the logger:

```
```csharp
...```

```

```

using Castle.Core.Logging; //1: Import Logging namespace

public class TaskAppService : ITaskAppService
{
 //2: Getting a logger using property injection
 public ILogger Logger { get; set; }

 public TaskAppService()
 {
 //3: Do not write logs if no Logger supplied.
 Logger = NullLogger.Instance;
 }

 public void CreateTask(CreateTaskInput input)
 {
 //4: Write logs
 Logger.Info("Creating a new task with description: " +
input.Description);
 //...
 }
}
```

```

ABP Framework depends on Microsoft's [logging extensions](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging>) library which is also an abstraction and there are many providers implement it. Startup templates are using the Serilog as the pre-configured logging library while it is easy to change in your project. The usage pattern is similar:

```

```csharp
//1: Import the Logging namespaces
using Microsoft.Extensions.Logging;
using Microsoft.Extensions.Logging.Abstractions;

public class TaskAppService : ITaskAppService
{
 //2: Getting a logger using property injection
 public ILogger<TaskAppService> Logger { get; set; }

 public TaskAppService()
 {
 //3: Do not write logs if no Logger supplied.
 Logger = NullLogger<TaskAppService>.Instance;
 }

 public void CreateTask(CreateTaskInput input)
 {
 //4: Write logs
 Logger.Info("Creating a new task with description: " +
input.Description);
 //...
 }
}
```

```

You inject the `ILogger<T>` instead of the `ILogger`.

```
### Object to Object Mapping
```

```
#### IObjectMapper Service
```

ASP.NET Boilerplate defines an `IObjectMapper` service ([see](<https://aspnetboilerplate.com/Pages/Documents/Object-To-Object-Mapping>)) and has an integration to the [AutoMapper](<https://automapper.org/>) library.

Example usage: Create a `User` object with the given `CreateUserInput` object:

```
```csharp
public void CreateUser(CreateUserInput input)
{
 var user = ObjectMapper.Map<User>(input);
 ...
}
...```

```

Example: Update an existing `User` properties with the given `UpdateUserInput` object:

```
```csharp
public async Task UpdateUserAsync(Guid id, UpdateUserInput input)
{
    var user = await _userRepository.GetAsync(id);
    ObjectMapper.Map(input, user);
}
...```

```

ABP Framework has the same `IObjectMapper` service ([see]([Object-To-Object-Mapping.md](#))) and the AutoMapper integration with a slightly different mapping methods.

Example usage: Create a `User` object with the given `CreateUserInput` object:

```
```csharp
public void CreateUser(CreateUserInput input)
{
 var user = ObjectMapper.Map<CreateUserInput, User>(input);
}
...```

```

This time you need to explicitly declare the source type and target type (while ASP.NET Boilerplate was requiring only the target type).

Example: Update an existing `User` properties with the given `UpdateUserInput` object:

```
```csharp
public async Task UpdateUserAsync(Guid id, UpdateUserInput input)
{
    var user = await _userRepository.GetAsync(id);
    ObjectMapper.Map<UpdateUserInput, User>(input, user);
}
...```

```

Again, ABP Framework expects to explicitly set the source and target types.

AutoMapper Integration

Auto Mapping Attributes

ASP.NET Boilerplate has `AutoMapTo`, `AutoMapFrom` and `AutoMap` attributes to automatically create mappings for the declared types. Example:

```
```csharp
[AutoMapTo(typeof(User))]
public class CreateUserInput
{
 public string Name { get; set; }
 public string Surname { get; set; }
 ...
}
````
```

ABP Framework has no such attributes, because AutoMapper as a [similar attribute](<https://automapper.readthedocs.io/en/latest/Attribute-mapping.html>) now. You need to switch to AutoMapper's attribute.

Mapping Definitions

ABP Framework follows AutoMapper principles closely. You can define classes derived from the `Profile` class to define your mappings.

Configuration Validation

Configuration validation is a best practice for the AutoMapper to maintain your mapping configuration in a safe way.

See [the documentation](Object-To-Object-Mapping.md) for more information related to the object mapping.

Setting Management

Defining the Settings

In an ASP.NET Boilerplate based application, you create a class deriving from the `SettingProvider` class, implement the `GetSettingDefinitions` method and add your class to the `Configuration.Settings.Providers` list.

In the ABP Framework, you need to derive your class from the `SettingDefinitionProvider` and implement the `Define` method. You don't need to register your class since the ABP Framework automatically discovers it.

Getting the Setting Values

ASP.NET Boilerplate provides the `ISettingManager` to read the setting values in the server side and `abp.setting.get(...)` method in the JavaScript side.

ABP Framework has the `ISettingProvider` service to read the setting values in the server side and `abp.setting.get(...)` method in the JavaScript side.

Setting the Setting Values

For ASP.NET Boilerplate, you use the same '`ISettingManager`' service to change the setting values.

ABP Framework separates it and provides the setting management module (pre-added to the startup projects) which has the '`ISettingManager`' to change the setting values. This separation was introduced to support tiered deployment scenarios (where '`ISettingProvider`' can also work in the client application while '`ISettingManager`' can also work in the server (API) side).

Clock

ASP.NET Boilerplate has a static '`Clock`' service ([see](<https://aspnetboilerplate.com/Pages/Documents/Timing>)) which is used to abstract the '`DateTime`' kind, so you can easily switch between Local and UTC times. You don't inject it, but just use the '`Clock.Now`' static method to obtain the current time.

ABP Framework has the '`IClock`' service ([see](Timing.md)) which has a similar goal, but now you need to inject it whenever you need it.

Event Bus

ASP.NET Boilerplate has an in-process event bus system. You typically inject the '`IEventBus`' (or use the static instance '`EventBus.Default`') to trigger an event. It automatically triggers events for entity changes (like '`EntityCreatingEventData`' and '`EntityUpdatedEventData`'). You create a class by implementing the '`IEventHandler<T>`' interface.

ABP Framework separates the event bus into two services: '`ILocalEventBus`' and '`IDistributedEventBus`'.

The local event bus is similar to the event bus of the ASP.NET Boilerplate while the distributed event bus is new feature introduced in the ABP Framework.

So, to migrate your code;

- * Use the '`ILocalEventBus`' instead of the '`IEventBus`'.
- * Implement the '`ILocalEventHandler`' instead of the '`IEventHandler`'.

> Note that ABP Framework has also an '`IEventBus`' interface, but it does exists to be a common interface for the local and distributed event bus. It is not injected and directly used.

Feature Management

Feature system is used in multi-tenant applications to define features of your application check if given feature is available for the current tenant.

Defining Features

In the ASP.NET Boilerplate ([see](<https://aspnetboilerplate.com/Pages/Documents/Feature-Management>))), you create a class inheriting from the '`FeatureProvider`', override the '`SetFeatures`' method and add your class to the '`Configuration.Features.Providers`' list.

In the ABP Framework ([see](Features.md)), you derive your class from the `FeatureDefinitionProvider` and override the `Define` method. No need to add your class to the configuration, it is automatically discovered by the framework.

Checking Features

You can continue to use the `RequiresFeature` attribute and `IFeatureChecker` service to check if a feature is enabled for the current tenant.

Changing the Feature Values

In the ABP Framework you use the `IFeatureManager` to change a feature value for a tenant.

Audit Logging

The ASP.NET Boilerplate ([see](https://aspnetboilerplate.com/Pages/Documents/Audit-Logging)) and the ABP Framework ([see](Audit-Logging.md)) has similar audit logging systems. ABP Framework requires to add `UseAuditing()` middleware to the ASP.NET Core pipeline, which is already added in the startup templates. So, most of the times it will be work out of the box.

Localization

ASP.NET Boilerplate supports XML and JSON files to define the localization key-values for the UI ([see](https://aspnetboilerplate.com/Pages/Documents/Localization)). ABP Framework only supports the JSON formatter localization files ([see](Localization.md)). So, you need to convert your XML file to JSON.

The ASP.NET Boilerplate has its own the `ILocalizationManager` service to be injected and used for the localization in the server side.

The ABP Framework uses [Microsoft localization extension](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization) library, so it is completely integrated to ASP.NET Core. You use the `IStringLocalizer<T>` service to get a localized text. Example:

```
```csharp
public class MyService
{
 private readonly IStringLocalizer<TestResource> _localizer;

 public MyService(IStringLocalizer<TestResource> localizer)
 {
 _localizer = localizer;
 }

 public void Foo()
 {
 var str = _localizer["HelloWorld"]; //Get a localized text
 }
}
```

```

So, you need to replace `ILocalizationManager` usage by the `IStringLocalizer`.

It also provides API used in the client side:

```
```js
var testResource = abp.localization.getResource('Test');
var str = testResource('HelloWorld');
````
```

It was like `abp.localization.localize(...)` in the ASP.NET Boilerplate.

Navigation vs Menu

In ASP.NET Boilerplate you create a class deriving from the `NavigationProvider` to define your menu elements. Menu items has `requiredPermissionName` attributes to restrict access to a menu element. Menu items were static and your class is executed only one time.

In the ABP Framework you need to create a class implements the `IMenuContributor` interface. Your class is executed whenever the menu needs to be rendered. So, you can conditionally add menu items.

As an example, this is the menu contributor of the tenant management module:

```
```csharp
public class AbpTenantManagementWebMainMenuContributor : IMenuContributor
{
 public async Task ConfigureMenuAsync(MenuConfigurationContext context)
 {
 //Add items only to the main menu
 if (context.Menu.Name != StandardMenus.Main)
 {
 return;
 }

 //Get the standard administration menu item
 var administrationMenu = context.Menu.GetAdministration();

 //Resolve some needed services from the DI container
 var l = context.GetLocalizer<AbpTenantManagementResource>();

 var tenantManagementMenuItem = new ApplicationMenuItem(
 TenantManagementMenuNames.GroupName,
 l["Menu:TenantManagement"],
 icon: "fa fa-users");

 administrationMenu.AddItem(tenantManagementMenuItem);

 //Conditionally add the "Tenants" menu item based on the permission
 if (await
context.IsGrantedAsync(TenantManagementPermissions.Tenants.Default))
 {
 tenantManagementMenuItem.AddItem(
 new ApplicationMenuItem(
 TenantManagementMenuNames.Tenants,
 l["Tenants"],
 url: "/TenantManagement/Tenants"));
 }
 }
}
```

```
 }
 }
}...
```

So, you need to check permission using the `IAuthorizationService` if you want to show a menu item only when the user has the related permission.

> Navigation/Menu system is only for ASP.NET Core MVC / Razor Pages applications. Angular applications has a different system implemented in the startup templates.

#### ## Missing Features

The following features are not present for the ABP Framework. Here, a list of some major missing features (and the related issue for that feature waiting on the ABP Framework GitHub repository):

- \* [Multi-Lingual Entities](<https://aspnetboilerplate.com/Pages/Documents/Multi-Lingual-Entities>) ([#1754](<https://github.com/abpframework/abp/issues/1754>))
- \* [Real time notification system](<https://aspnetboilerplate.com/Pages/Documents/Notification-System>) ([#633](<https://github.com/abpframework/abp/issues/633>))
- \* [NHibernate Integration](<https://aspnetboilerplate.com/Pages/Documents/NHibernate-Integration>) ([#339](<https://github.com/abpframework/abp/issues/339>)) – We don't intent to work on this, but any community contribution welcome.

Some of these features will eventually be implemented. However, you can implement them yourself if they are important for you. If you want, you can [contribute](Contribution/Index.md) to the framework, it is appreciated.

## 4 CLI

#### # ABP CLI

ABP CLI (Command Line Interface) is a command line tool to perform some common operations for ABP based solutions.

#### ## Installation

ABP CLI is a [dotnet global tool](<https://docs.microsoft.com/en-us/dotnet/core/tools/global-tools>). Install it using a command line window:

```
```bash
dotnet tool install -g Volo.Abp.Cli
```
```

To update an existing installation:

```
```bash
dotnet tool update -g Volo.Abp.Cli
```
```

## ## Global Options

While each command may have a set of options, there are some global options that can be used with any command;

\* `--skip-cli-version-check`: Skips to check the latest version of the ABP CLI. If you don't specify, it will check the latest version and shows a warning message if there is a newer version of the ABP CLI.

## ## Commands

Here, is the list of all available commands before explaining their details:

- \* \*\*`help`\*\*: Shows help on the usage of the ABP CLI.
- \* \*\*`cli`\*\*: Update or remove ABP CLI.
- \* \*\*`new`\*\*: Generates a new solution based on the ABP [startup templates](Startup-Templates/Index.md).
- \* \*\*`update`\*\*: Automatically updates all ABP related NuGet and NPM packages in a solution.
- \* \*\*`clean`\*\*: Deletes all `BIN` and `OBJ` folders in the current folder.
- \* \*\*`add-package`\*\*: Adds an ABP package to a project.
- \* \*\*`add-module`\*\*: Adds a [multi-package application module](https://docs.abp.io/en/abp/latest/Modules/Index) to a solution.
- \* \*\*`list-modules`\*\*: Lists names of open-source application modules.
- \* \*\*`list-templates`\*\*: Lists the names of available templates to create a solution.
- \* \*\*`get-source`\*\*: Downloads the source code of a module.
- \* \*\*`generate-proxy`\*\*: Generates client side proxies to use HTTP API endpoints.
- \* \*\*`remove-proxy`\*\*: Removes previously generated client side proxies.
- \* \*\*`switch-to-preview`\*\*: Switches to the latest preview version of the ABP Framework.
- \* \*\*`switch-to-nightly`\*\*: Switches to the latest [nightly builds](Nightly-Builds.md) of the ABP related packages on a solution.
- \* \*\*`switch-to-stable`\*\*: Switches to the latest stable versions of the ABP related packages on a solution.
- \* \*\*`switch-to-local`\*\*: Changes NuGet package references on a solution to local project references.
- \* \*\*`translate`\*\*: Simplifies to translate localization files when you have multiple JSON [localization](Localization.md) files in a source control repository.
- \* \*\*`login`\*\*: Authenticates on your computer with your [abp.io](https://abp.io/) username and password.
- \* \*\*`login-info`\*\*: Shows the current user's login information.
- \* \*\*`logout`\*\*: Logouts from your computer if you've authenticated before.
- \* \*\*`bundle`\*\*: Generates script and style references for ABP Blazor and MAUI Blazor project.
- \* \*\*`install-libs`\*\*: Install NPM Packages for MVC / Razor Pages and Blazor Server UI types.
- \* \*\*`clear-download-cache`\*\*: Clears the templates download cache.

## ### help

Shows basic usages of the ABP CLI.

Usage:

```
```bash
```

```
abp help [command-name]
```

Examples:

```
```bash
abp help # Shows a general help.
abp help new # Shows help about the "new" command.
````
```

cli

Update or remove ABP CLI.

Usage:

```
```bash
abp cli [command-name]
````
```

Examples:

```
```bash
abp cli update
abp cli update --preview
abp cli update --version 5.0.0
abp cli remove
````
```

new

Generates a new solution based on the ABP [startup templates](Startup-Templates/Index.md).

Usage:

```
```bash
abp new <solution-name> [options]
````
```

Example:

```
```bash
abp new Acme.BookStore
````
```

* `Acme.BookStore` is the solution name here.
* Common convention is to name a solution like *YourCompany.YourProject*. However, you can use different naming like *YourProject* (single level namespacing) or *YourCompany.YourProduct.YourModule* (three levels namespacing).

For more samples, go to [ABP CLI Create Solution Samples](CLI-New-Command-Samples.md)

Options


```

* **`app-nolayers`**: [Single-layer application template](Startup-Templates/Application-Single-Layer.md). Additional options:
  * `--ui` or `-u`: Specifies the UI framework. Default framework is `mvc`.
Available frameworks:
  * `mvc`: ASP.NET Core MVC.
  * `angular`: Angular UI.
  * `blazor`: Blazor UI.
  * `blazor-server`: Blazor Server UI.
  * `none`: Without UI.
  * `--database-provider` or `-d`: Specifies the database provider. Default provider is `ef`. Available providers:
    * `ef`: Entity Framework Core.
    * `mongodb`: MongoDB.
  * `--theme`: Specifies the theme. Default theme is `leptonx-lite`.
Available themes:
  * `leptonx-lite`: [LeptonX Lite Theme](Themes/LeptonXLite/AspNetCore.md).
    * `basic`: [Basic Theme](UI/AspNetCore/Basic-Theme.md).
* **`maui`**: .NET MAUI. A minimalist .NET MAUI application will be created if you specify this option.
* `--output-folder` or `-o`: Specifies the output folder. Default value is the current directory.
* `--version` or `-v`: Specifies the ABP & template version. It can be a [release tag](https://github.com/abpframework/abp/releases) or a [branch name](https://github.com/abpframework/abp/branches). Uses the latest release if not specified. Most of the times, you will want to use the latest version.
* `--preview`: Use latest preview version.
* `--template-source` or `-ts`: Specifies a custom template source to use to build the project. Local and network sources can be used(Like `D:\local-template` or `https://.../my-template-file.zip`).
* `--create-solution-folder` or `-csf`: Specifies if the project will be in a new folder in the output folder or directly the output folder.
* `--connection-string` or `-cs`: Overwrites the default connection strings in all `appsettings.json` files. The default connection string is `Server=localhost;Database=MyProjectName;Trusted_Connection=True` for EF Core and it is configured to use the SQL Server. If you want to use the EF Core, but need to change the DBMS, you can change it as [described here](Entity-Framework-Core-Other-DBMS.md) (after creating the solution).
* `--database-management-system` or `-dbms`: Sets the database management system. Default is **SQL Server**. Supported DBMS's:
  * `SqlServer`
  * `MySQL`
  * `SQLite`
  * `Oracle`
  * `Oracle-Devart`
  * `PostgreSQL`
* `--local-framework-ref --abp-path`: Uses local projects references to the ABP framework instead of using the NuGet packages. This can be useful if you download the ABP Framework source code and have a local reference to the framework from your application.
* `--no-random-port`: Uses template's default ports.
* `--skip-installing-libs` or `-sib`: Skip installing client side packages.
* `--skip-cache` or `-sc`: Always download the latest from our server and refresh their templates folder cache.
* `--with-public-website`: **Public Website** is a front-facing website for describing your project, listing your products and doing SEO for marketing purposes. Users can login and register on your website with this website.

```

See some [examples for the new command](CLI-New-Command-Samples.md) here.

update

Updating all ABP related packages can be tedious since there are many packages of the framework and modules. This command automatically updates all ABP related NuGet and NPM packages in a solution or project to the latest versions. You can run it in the root folder of your solutions.

Usage:

```
```bash
abp update [options]
```
```

- * If you run in a directory with a .csproj file, it updates all ABP related packages of the project to the latest versions.
- * If you run in a directory with a .sln file, it updates all ABP related packages of the all projects of the solution to the latest versions.
- * If you run in a directory that contains multiple solutions in sub-folders, it can update all the solutions, including Angular projects.

Note that this command can upgrade your solution from a previous version, and also can upgrade it from a preview release to the stable release of the same version.

Options

- * `--npm`: Only updates NPM packages.
- * `--nuget`: Only updates NuGet packages.
- * `--solution-path` or `-sp`: Specify the solution path. Use the current directory by default
- * `--solution-name` or `-sn`: Specify the solution name. Search `*.sln` files in the directory by default.
- * `--check-all`: Check the new version of each package separately. Default is `false`.
- * `--version` or `-v`: Specifies the version to use for update. If not specified, latest version is used.

clean

Deletes all `BIN` and `OBJ` folders in the current folder.

Usage:

```
```bash
abp clean
```
```

add-package

Adds an ABP package to a project by,

- * Adding related nuget package as a dependency to the project.
- * Adding `[DependsOn(...)]` attribute to the module class in the project (see the [module development document](Module-Development-Basics.md)).

> Notice that the added module may require additional configuration which is generally indicated in the documentation of the related package.

Basic usage:

```
```bash
abp add-package <package-name> [options]
````
```

Example:

```
```
abp add-package Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic
````
```

* This example adds the `Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic` package to the project.

Options

- * `--project` or `-p`: Specifies the project (.csproj) file path. If not specified, CLI tries to find a .csproj file in the current directory.
- * `--with-source-code`: Downloads the source code of the package to your solution folder and uses local project references instead of NuGet/NPM packages.
- * `--add-to-solution-file`: Adds the downloaded package to your solution file, so you will also see the package when you open the solution on a IDE. (only available when `--with-source-code` is True)

> Currently only the source code of the basic theme packages([MVC](https://docs.abp.io/en/abp/latest/UI/AspNetCore/Basic-Theme) and [Blazor](https://docs.abp.io/en/abp/latest/UI/Blazor/Basic-Theme)) can be downloaded.

- > - Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic
- > - Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme
- > - Volo.Abp.AspNetCore.Components.Web.BasicTheme
- > - Volo.Abp.AspNetCore.Components.Server.BasicTheme

add-module

Adds a [multi-package application module](Modules/Index) to a solution by finding all packages of the module, finding related projects in the solution and adding each package to the corresponding project in the solution.

It can also create a new module for your solution and add it to your solution. See `--new` option.

> A business module generally consists of several packages (because of layering, different database provider options or other reasons). Using `add-module` command dramatically simplifies adding a module to a solution. However, each module may require some additional configurations which is generally indicated in the documentation of the related module.

Usage:

```
```bash
abp add-module <module-name> [options]
````
```

....

Examples:

```
```bash
abp add-module Volo.Blogging
````
```

* This example adds the `Volo.Blogging` module to the solution.

```
```bash
abp add-module ProductManagement --new --add-to-solution-file
````
```

* This command creates a fresh new module customized for your solution (named `ProductManagement`) and adds it to your solution.

Options

- * `--solution` or `-s`: Specifies the solution (.sln) file path. If not specified, CLI tries to find a .sln file in the current directory.
- * `--skip-db-migrations`: For EF Core database provider, it automatically adds a new code first migration (`Add-Migration`) and updates the database (`Update-Database`) if necessary. Specify this option to skip this operation.
- * `-sp` or `--startup-project`: Relative path to the project folder of the startup project. Default value is the current folder.
- * `--new`: Creates a fresh new module (customized for your solution) and adds it to your solution.
- * `--with-source-code`: Downloads the source code of the module to your solution folder and uses local project references instead of NuGet/NPM packages. This option is always `True` if `--new` is used.
- * `--add-to-solution-file`: Adds the downloaded/created module to your solution file, so you will also see the projects of the module when you open the solution on a IDE. (only available when `--with-source-code` is `True`.)

list-modules

Lists names of open-source application modules.

Usage:

```
```bash
abp list-modules [options]
````
```

Example:

```
```bash
abp list-modules
````
```

Options

- * `--include-pro-modules`: Includes commercial (pro) modules in the output.

list-templates

Lists all available templates to create a solution.

Usage:

```
```bash
abp list-templates
````
```

get-source

Downloads the source code of a module to your computer.

Usage:

```
```bash
abp get-source <module-name> [options]
````
```

Example:

```
```bash
abp get-source Volo.Blogging

abp get-source Volo.Blogging --local-framework-ref --abp-path D:\GitHub\abp
````
```

Options

- * `--output-folder` or `-o`: Specifies the directory that source code will be downloaded in. If not specified, current directory is used.
- * `--version` or `-v`: Specifies the version of the source code that will be downloaded. If not specified, latest version is used.
- * `--preview`: If no version option is specified, this option specifies if latest [preview version](Previews.md) will be used instead of latest stable version.
- * `--local-framework-ref --abp-path`: Path of [ABP Framework GitHub repository](https://github.com/abpframework/abp) in your computer. This will be used for converting project references to your local system. If this is not specified, project references will be converted to NuGet references.

generate-proxy

Generates Angular, C# or JavaScript service proxies for your HTTP APIs to make easy to consume your services from the client side. Your host (server) application must be up and running before running this command.

Usage:

```
```bash
abp generate-proxy -t <client-type> [options]
````
```

Examples:

```
```bash
abp generate-proxy -t ng -url https://localhost:44302/
abp generate-proxy -t js -url https://localhost:44302/
abp generate-proxy -t csharp -url https://localhost:44302/
````
```

....

Options

- * `--type` or `-t`: The name of client type. Available clients:
 - * `csharp`: C#, work in the `*.HttpApi.Client` project directory. There are some additional options for this client:
 - * `--without-contracts`: Avoid generating the application service interface, class, enum and dto types.
 - * `--folder`: Folder name to place generated CSharp code in. Default value: `ClientProxies`.
 - * `ng`: Angular. There are some additional options for this client:
 - * `--api-name` or `-a`: The name of the API endpoint defined in the `/src/environments/environment.ts`. Default value: `default`.
 - * `--source` or `-s`: Specifies the Angular project name to resolve the root namespace & API definition URL from. Default value: `defaultProject`.
 - * `--target`: Specifies the Angular project name to place generated code in. Default value: `defaultProject`.
 - * `--module`: Backend module name. Default value: `app`.
 - * `--entry-point`: Targets the Angular project to place the generated code.
 - * `--url`: Specifies api definition url. Default value is API Name's url in environment file.
 - * `--prompt` or `-p`: Asks the options from the command line prompt (for the unspecified options).
 - * `js`: JavaScript. work in the `*.Web` project directory. There are some additional options for this client:
 - * `--output` or `-o`: JavaScript file path or folder to place generated code in.
 - * `--module` or `-m`: Specifies the name of the backend module you wish to generate proxies for. Default value: `app`.
 - * `--working-directory` or `-wd`: Execution directory. For `csharp` and `js` client types.
 - * `--url` or `-u`: API definition URL from.
 - * `--service-type` or `st`: Specifies the service type to generate. `application`, `integration` and `all`, Default value: `all` for C#, `application` for JavaScript / Angular.

> See the [\[Angular Service Proxies document\]](#)(UI/Angular/Service-Proxies.md) for more.

remove-proxy

Removes previously generated proxy code from the Angular, CSharp or JavaScript application. Your host (server) application must be up and running before running this command.

This can be especially useful when you generate proxies for multiple modules before and need to remove one of them later.

Usage:

```
```bash
abp remove-proxy -t <client-type> [options]
````
```

Examples:

```
```bash
abp remove-proxy -t ng
abp remove-proxy -t js -m identity -o Pages/Identity/client-proxies.js
abp remove-proxy -t csharp --folder MyProxies/InnerFolder
````
```

Options

- * `--type` or `-t`: The name of client type. Available clients:
 - * `csharp`: C#, work in the `*.HttpApiClient` project directory. There are some additional options for this client:
 - * `--folder`: Folder name to place generated CSharp code in. Default value: `ClientProxies`.
 - * `ng`: Angular. There are some additional options for this client:
 - * `--api-name` or `-a`: The name of the API endpoint defined in the `/src/environments/environment.ts`. Default value: `default`.
 - * `--source` or `-s`: Specifies the Angular project name to resolve the root namespace & API definition URL from. Default value: `defaultProject`.
 - * `--target`: Specifies the Angular project name to place generated code in. Default value: `defaultProject`.
 - * `--url`: Specifies api definition url. Default value is API Name's url in environment file.
 - * `--prompt` or `-p`: Asks the options from the command line prompt (for the unspecified options).
 - * `js`: JavaScript. work in the `*.Web` project directory. There are some additional options for this client:
 - * `--output` or `-o`: JavaScript file path or folder to place generated code in.
 - * `--module` or `-m`: Specifies the name of the backend module you wish to generate proxies for. Default value: `app`.
 - * `--working-directory` or `-wd`: Execution directory. For `csharp` and `js` client types.
 - * `--url` or `-u`: API definition URL from.

> See the [Angular Service Proxies document](UI/Angular/Service-Proxies.md) for more.

switch-to-preview

You can use this command to switch your solution or project to latest preview version of the ABP framework.

Usage:

```
```bash
abp switch-to-preview [options]
````
```

Options

- * `--directory` or `-d`: Specifies the directory. The solution or project should be in that directory or in any of its sub directories. If not specified, default is the current directory.

switch-to-nightly

You can use this command to switch your solution or project to latest [nightly](Nightly-Builds.md) preview version of the ABP framework packages.

Usage:

```
```bash
abp switch-to-nightly [options]
```
```

Options

- * `--directory` or `-d`: Specifies the directory. The solution or project should be in that directory or in any of its sub directories. If not specified, default is the current directory.

switch-to-stable

If you're using the ABP Framework preview packages (including nightly previews), you can switch back to latest stable version using this command.

Usage:

```
```bash
abp switch-to-stable [options]
```
```

Options

- * `--directory` or `-d`: Specifies the directory. The solution or project should be in that directory or in any of its sub directories. If not specified, default is the current directory.

switch-to-local

Changes all NuGet package references to local project references for all the .csproj files in the specified folder (and all its subfolders with any deep). It is not limited to ABP Framework or Module packages.

Usage:

```
```bash
abp switch-to-local [options]
```
```

Options

- * `--solution` or `-s`: Specifies the solution directory. The solution should be in that directory or in any of its sub directories. If not specified, default is the current directory.

- * `--paths` or `-p`: Specifies the local paths that the projects are inside. You can use `|` character to separate the paths.

Example:

```
```bash
abp switch-to-local --paths "D:\Github\abp|D:\Github\my-repo"
```
```

translate

Simplifies to translate [localization](Localization.md) files when you have multiple JSON [localization](Localization.md) files in a source control repository.

- * This command will create a unified json file based on the reference culture.
 - * It searches all the localization 'JSON' files in the current directory and all subdirectories (recursively). Then creates a single file (named 'abp-translation.json' by default) that includes all the entries need to be translated.
 - * Once you translate the entries in this file, you can then apply your changes to the original localization files using the '--apply' command.
- > The main purpose of this command is to translate ABP Framework localization files (since the [abp repository](https://github.com/abpframework/abp) has tens of localization files to be translated in different directories).

Creating the Translation File

First step is to create the unified translation file:

```
```bash
abp translate -c <culture> [options]
````
```

Example:

```
```bash
abp translate -c de
````
```

This command created the unified translation file for the 'de' (German) culture.

Additional Options

- * '--reference-culture' or '-r': Default 'en'. Specifies the reference culture.
- * '--output' or '-o': Output file name. Default 'abp-translation.json'.
- * '--all-values' or '-all': Include all keys to translate. By default, the unified translation file only includes the missing texts for the target culture. Specify this parameter if you may need to revise the values already translated before.

Applying Changes

Once you translate the entries in the unified translation file, you can apply your changes to the original localization files using the '--apply' parameter:

```
```bash
abp translate --apply # apply all changes
abp translate -a # shortcut for --apply
````
```

Then review changes on your source control system to be sure that it has changed the proper files and send a Pull Request if you've translated ABP Framework resources. Thank you in advance for your contribution.

Additional Options

* `--file` or `-f`: Default: `abp-translation.json`. The translation file (use only if you've used the `--output` option before).

login

Some features of the CLI requires to be logged in to abp.io platform. To login with your username write:

```
```bash
abp login <username> # Allows you to enter
your password hidden
abp login <username> -p <password> # Specify the password
as a parameter (password is visible)
abp login <username> --organization <organization> # If you have multiple
organizations, you need set your active organization
abp login <username> -p <password> -o <organization> # You can enter both
your password and organization in the same command
abp login <username> --device # Use device login flow
```

```

> When using the -p parameter, be careful as your password will be visible. It's useful for CI/CD automation pipelines.

A new login with an already active session overwrites the previous session.

login-info

Shows your login information such as **Name**, **Surname**, **Username**, **Email Address** and **Organization**.

```
```bash
abp login-info
```

```

logout

Logs you out by removing the session token from your computer.

```
```bash
abp logout
```

```

bundle

This command generates script and style references for ABP Blazor WebAssembly and MAUI Blazor project and updates the **index.html** file. It helps developers to manage dependencies required by ABP modules easily. In order `bundle` command to work, its **executing directory** or passed `--working-directory` parameter's directory must contain a Blazor or MAUI Blazor project file(*.csproj).

Usage:

```
```bash
abp bundle [options]
```

#### Options

* ``--working-directory`` or ``-wd``: Specifies the working directory. This option is useful when executing directory doesn't contain a Blazor project file.
* ``--force`` or ``-f``: Forces to build project before generating references.
* ``--project-type`` or ``-t``: Specifies the project type. Default type is `webassembly`. Available types:
  * `webassembly`
  * `maui-blazor`
```

`bundle` command reads the `appsettings.json` file inside the Blazor and MAUI Blazor project for bundling options. For more details about managing style and script references in Blazor or MAUI Blazor apps, see [\[Managing Global Scripts & Styles\]\(UI/Blazor/Global-Scripts-Styles.md\)](#)

install-libs

This command install NPM Packages for MVC / Razor Pages and Blazor Server UI types. Its **executing directory** or passed ``--working-directory`` parameter's directory must contain a project file(*.csproj).

`install-libs` command reads the `abp.resourcemapping.js` file to manage package. For more details see [\[Client Side Package Management\]\(UI/AspNetCore/Client-Side-Package-Management.md\)](#).

Usage:

```
```bash
abp install-libs [options]
```
```

Options

* ``--working-directory`` or ``-wd``: Specifies the working directory. This option is useful when executing directory doesn't contain a project file.

See Also

* [\[Examples for the new command\]\(CLI-New-Command-Samples.md\)](#)

4.1 Examples for the new command

ABP CLI – New Solution Sample Commands

The `abp new` command creates an ABP solution or other artifacts based on an ABP template. [\[ABP CLI\]\(CLI.md\)](#) has several parameters to create a new ABP solution. In this document we will show you some sample commands to create a new solution. All the project names are `Acme.BookStore`. Currently, the

available mobile projects are 'React Native' and 'MAUI' mobile app. Available database providers are 'Entity Framework Core' and 'MongoDB'. All the commands starts with 'abp new'.

Angular

The following commands are for creating Angular UI projects:

* **Entity Framework Core**, no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u angular --mobile none --database-provider ef -csf
````
```

* **Entity Framework Core**, default app template, **separate Auth Server**, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -t app -u angular -m none --separate-auth-server --
database-provider ef -csf
````
```

* **Entity Framework Core**, **custom connection string**, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u angular -csf --connection-string
Server=localhost;Database=MyDatabase;Trusted_Connection=True
````
```

* **MongoDB**, default app template, mobile project included, creates solution in 'C:\MyProjects\Acme.BookStore'

```
```bash
abp new Acme.BookStore -u angular --database-provider mongodb --output-
folder C:\MyProjects\Acme.BookStore
````
```

* **MongoDB**, default app template, no mobile app, **separate Auth Server**, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -t app -u angular -m none --separate-auth-server --
database-provider mongodb -csf
````
```

MVC

The following commands are for creating MVC UI projects:

* **Entity Framework Core**, no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -t app -u mvc --mobile none --database-provider ef -
csf
````
```

* **Entity Framework Core**, **tier architecture** (**Web and HTTP API are separated**), no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u mvc --mobile none --tiered --database-provider ef
-csf
````
```

* **MongoDB**, no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -t app -u mvc --mobile none --database-provider
mongodb -csf
````
```

* **MongoDB**, **tier architecture**, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u mvc --tiered --database-provider mongodb -csf
````
```

* **Public Website**, Entity Framework Core, no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -t app -u mvc --mobile none --database-provider ef -
csf --with-public-website
````
```

Note that Public Website is only included in PRO templates..

Blazor WebAssembly

The following commands are for creating Blazor WASM projects:

* **Entity Framework Core**, no mobile app:

```
```bash
abp new Acme.BookStore -t app -u blazor --mobile none
````
```

* **Entity Framework Core**, **separate Auth Server**, mobile app included:

```
```bash
abp new Acme.BookStore -u blazor --separate-auth-server
````
```

* **MongoDB**, no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u blazor --database-provider mongodb --mobile none
-csf
````
```

Blazor Server

The following commands are for creating Blazor projects:

* **Entity Framework Core**, no mobile app:

```
```bash
abp new Acme.BookStore -t app -u blazor-server --mobile none
````
```

* **Entity Framework Core**, **separate Auth Server**, **separate API Host**, mobile app included:

```
```bash
abp new Acme.BookStore -u blazor-server --tiered
````
```

* **MongoDB**, no mobile app, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u blazor --database-provider mongodb --mobile none
--csf
````
```

No UI

In the default app template, there is always a frontend project. In this option there is no frontend project. It has a `HttpApi.Host` project to serve your HTTP WebAPIs. It's appropriate if you want to create a WebAPI service.

* **Entity Framework Core**, separate Auth Server, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u none --separate-auth-server --csf
````
```

* **MongoDB**, no mobile app:

```
```bash
abp new Acme.BookStore -u none --mobile none --database-provider mongodb
````
```

Console application

It's a template of a basic .NET console application with ABP module architecture integrated. To create a console application use the following command:

* This project consists of the following files: `Acme.BookStore.csproj`, `appsettings.json`, `BookStoreHostedService.cs`, `BookStoreModule.cs`, `HelloWorldService.cs` and `Program.cs`.

```
```bash
abp new Acme.BookStore -t console --csf
````
```

Module

Module are reusable sub applications used by your main project. Using ABP Module is a best practice if you are building a microservice solution. As modules are not final applications, each module has all the frontend UI projects and database providers. The module template comes with an MVC UI to be able to develop without the final solution. But if you will develop your module under a final solution, you add `--no-ui` parameter to exclude MVC UI project.

* Included frontends: `MVC`, `Angular`, `Blazor`. Included database providers: `Entity Framework Core`, `MongoDB`. Includes MVC startup project.

```
```bash
abp new Acme.IssueManagement -t module
```

\* The same with the upper but doesn't include MVC startup project.

```
```bash
abp new Acme.IssueManagement -t module --no-ui
```

* Creates the module and adds it to your solution

```
```bash
abp new Acme.IssueManagement -t module --add-to-solution-file
```

#### ## Create a solution from a specific version

When you create a solution, it always creates with the latest version. To create a project from an older version, you can pass the `--version` parameter.

\* Create a solution from v3.3.0, with Angular UI and Entity Framework Core.

```
```bash
abp new Acme.BookStore -t app -u angular -m none --database-provider ef -
csf --version 3.3.0
````
```

To get the ABP version list, checkout following link:  
<https://www.nuget.org/packages/Volo.Abp.Core/>

#### ## Create from a custom template

ABP CLI uses the default [app template](<https://github.com/abpframework/abp/tree/dev/templates/app>) to create your project. If you want to create a new solution from your customized template, you can use the parameter `--template-source`.

\* MVC UI, Entity Framework Core, no mobile app, using the template in `c:\MyProjects\templates\app` directory.

```
```bash
abp new Acme.BookStore -t app -u mvc --mobile none --database-provider ef -
-template-source "c:\MyProjects\templates\app"
````
```

\* Same with the previous one except this command retrieves the template from the URL '<https://myabp.com/app-template.zip>'.

```
```bash
abp new Acme.BookStore -t app -u mvc --mobile none --database-provider ef --
-template-source https://myabp.com/app-template.zip
````
```

#### ## Create a preview version

ABP CLI always uses the latest version. In order to create a solution from a preview (RC) version add the '--preview' parameter.

\* Blazor UI, Entity Framework Core, no mobile, \*\*preview version\*\*, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -t app -u blazor --mobile none -csf --preview
````
```

#### ## Choose database management system

The default database management system (DBMS) is 'Entity Framework Core' / 'SQL Server'. You can choose a DBMS by passing '--database-management-system' parameter. [Accepted values](<https://github.com/abpframework/abp/blob/dev/framework/src/Volo.Abp.Cli.Core/Volo/Abp/Cli/ProjectBuilding/Building/DatabaseManagementSystem.cs>) are 'SqlServer', 'MySQL', 'SQLite', 'Oracle', 'Oracle-Devart', 'PostgreSQL'. The default value is 'SqlServer'.

\* Angular UI, \*\*PostgreSQL\*\* database, creates the project in a new folder:

```
```bash
abp new Acme.BookStore -u angular --database-management-system PostgreSQL -
csf
````
```

#### ## Use static HTTP ports

ABP CLI always assigns random ports to the hostable projects. If you need to keep the default ports and create a solution always with the same HTTP ports, add the parameter '--no-random-port'.

\* MVC UI, Entity Framework Core, \*\*static ports\*\*, creates the project in a new folder:

```
```bash
abp new Acme.BookStore --no-random-port -csf
````
```

#### ## Use local ABP framework references

ABP libraries are referenced from NuGet by default in the ABP solutions. Sometimes you need to reference ABP libraries locally to your solution. This is useful to debug the framework itself. Your local ABP Framework's root directory must have the 'Volo.Abp.sln' file. You can copy the content of the

```

following directory to your file system
https://github.com/abpframework/abp/tree/dev/framework

* MVC UI, Entity Framework Core, **ABP libraries are local project references**:

The local path must be the root directory of ABP repository.
If 'C:\source\abp\framework\Volo.Abp.sln' is your framework solution path,
then you must write 'C:\source\abp' to the '--abp-path' parameter.

```bash
abp new Acme.BookStore --local-framework-ref --abp-path C:\source\abp
```

Output:

As seen below, ABP Framework libraries are local project references.

```xml
<ItemGroup>
    <ProjectReference
        Include="C:\source\abp\framework\src\Volo.Abp.Autofac\Volo.Abp.Autofac.csproj"
    />
    <ProjectReference
        Include="C:\source\abp\framework\src\Volo.Abp.AspNetCore.Serilog\Volo.Abp.AspNetCore.Serilog.csproj" />
    <ProjectReference
        Include="C:\source\abp\framework\src\Volo.Abp.AspNetCore.Authentication.JwtBearer\Volo.Abp.AspNetCore.Authentication.JwtBearer.csproj" />
    <ProjectReference
        Include="..\Acme.BookStore.Application\Acme.BookStore.Application.csproj" />
    <ProjectReference
        Include="..\Acme.BookStore.HttpApi\Acme.BookStore.HttpApi.csproj" />
    <ProjectReference
        Include="..\Acme.BookStore.EntityFrameworkCore\Acme.BookStore.EntityFrameworkCore.csproj" />
</ItemGroup>
```

See Also

* [ABP CLI documentation](CLI.md)

```

## 5 Startup Templates

### 5.1 Overall

#### # Startup Templates

While you can start with an empty project and add needed packages manually, startup templates make easy and comfortable to start a new solution with the ABP framework. Click the name from the list below to see the documentation of the related startup template:

- \* [\*\*`app`\*\*](Application.md): Application template.

- \* [\*\*`app-nolayers`\*\*](Application-Single-Layer.md): Application (single layer) template.
- \* [\*\*`module`\*\*](Module.md): Module/service template.
- \* [\*\*`console`\*\*](Console.md): Console template.
- \* [\*\*`WPF`\*\*](WPF.md): WPF template.
- \* [\*\*`MAUI`\*\*](MAUI.md): MAUI template.

## 5.2 Application

# Application Startup Template

### ## Introduction

This template provides a layered application structure based on the [Domain Driven Design](../Domain-Driven-Design.md) (DDD) practices.

This document explains \*\*the solution structure\*\* and projects in details. If you want to start quickly, follow the guides below:

- \* [The getting started document](../Getting-Started.md) explains how to create a new application in a few minutes.
- \* [The application development tutorial](../Tutorials/Part-1) explains step by step application development.

### ## How to Start With?

You can use the [ABP CLI](../CLI.md) to create a new project using this startup template. Alternatively, you can generate a CLI command from the [Get Started](https://abp.io/get-started) page. CLI approach is used here.

First, install the ABP CLI if you haven't installed it before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
```
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.BookStore -t app
```
```

- \* `Acme.BookStore` is the solution name, like \*YourCompany.YourProduct\*. You can use single-level, two-level or three-level naming.
- \* This example specified the template name (`-t` or `--template` option). However, `app` is already the default template if you didn't specify it.

### ### Specify the UI Framework

This template provides multiple UI frameworks:

- \* `mvc`: ASP.NET Core MVC UI with Razor Pages (default)
- \* `blazor`: Blazor UI
- \* `blazor-server`: Blazor Server UI
- \* `angular`: Angular UI

Use the `-u` or `--ui` option to specify the UI framework:

```
```bash
abp new Acme.BookStore -u angular
````
```

### ### Specify the Database Provider

This template supports the following database providers:

- `ef`: Entity Framework Core (default)
- `mongodb`: MongoDB

Use `-d` (or `--database-provider`) option to specify the database provider:

```
```bash
abp new Acme.BookStore -d mongodb
````
```

### ### Specify the Mobile Application Framework

This template supports the following mobile application frameworks:

- `react-native`: React Native

Use the `-m` (or `--mobile`) option to specify the mobile application framework:

```
```bash
abp new Acme.BookStore -m react-native
````
```

If not specified, no mobile application will be created.

## ## Solution Structure

Based on the options you've specified, you will get a slightly different solution structure.

### ### Default Structure

If you don't specify any additional options, you will have a solution as shown below:

```
![bookstore-rider-solution-v6](../images/solution-structure-solution-explorer-rider.png)
```

Projects are organized in `src` and `test` folders. `src` folder contains the actual application which is layered based on [DDD](../Domain-Driven-Design.md) principles as mentioned before.

The diagram below shows the layers & project dependencies of the application:

```
![layered-project-dependencies](../images/layered-project-dependencies.png)
```

Each section below will explain the related project & its dependencies.

### #### .Domain.Shared Project

This project contains constants, enums and other objects these are actually a part of the domain layer, but needed to be used by all layers/projects in the solution.

A `BookType` enum and a `BookConsts` class (which may have some constant fields for the `Book` entity, like `MaxNameLength`) are good candidates for this project.

\* This project has no dependency on other projects in the solution. All other projects depend on this one directly or indirectly.

#### #### .Domain Project

This is the domain layer of the solution. It mainly contains [entities, aggregate roots](../Entities.md), [domain services](../Domain-Services.md), [value objects](../Value-Objects.md), [repository interfaces](../Repositories.md) and other domain objects.

A `Book` entity, a `BookManager` domain service and an `IBookRepository` interface are good candidates for this project.

\* Depends on the `.Domain.Shared` because it uses constants, enums and other objects defined in that project.

#### #### .Application.Contracts Project

This project mainly contains [application service](../Application-Services.md) \*\*interfaces\*\* and [Data Transfer Objects](../Data-Transfer-Objects.md) (DTO) of the application layer. It exists to separate the interface & implementation of the application layer. In this way, the interface project can be shared to the clients as a contract package.

An `IBookAppService` interface and a `BookCreationDto` class are good candidates for this project.

\* Depends on the `.Domain.Shared` because it may use constants, enums and other shared objects of this project in the application service interfaces and DTOs.

#### #### .Application Project

This project contains the [application service](../Application-Services.md) \*\*implementations\*\* of the interfaces defined in the `.Application.Contracts` project.

A `BookAppService` class is a good candidate for this project.

\* Depends on the `.Application.Contracts` project to be able to implement the interfaces and use the DTOs.

\* Depends on the `.Domain` project to be able to use domain objects (entities, repository interfaces... etc.) to perform the application logic.

#### #### .EntityFrameworkCore Project

This is the integration project for the EF Core. It defines the `DbContext` and implements repository interfaces defined in the `.Domain` project.

\* Depends on the `.[Domain](#)` project to be able to reference to entities and repository interfaces.

> This project is available only if you are using EF Core as the database provider. If you select another database provider, its name will be different.

#### #### .DbMigrator Project

This is a console application that simplifies the execution of database migrations on development and production environments. When you run this application, it:

- \* Creates the database if necessary.
- \* Applies the pending database migrations.
- \* Seeds initial data if needed.

> This project has its own `appsettings.json` file. So, if you want to change the database connection string, remember to change this file too.

Especially, seeding initial data is important at this point. ABP has a modular data seed infrastructure. See [[its documentation](#)](./Data-Seeding.md) for more about the data seeding.

While creating database & applying migrations seem only necessary for relational databases, this project comes even if you choose a NoSQL database provider (like MongoDB). In that case, it still seeds the initial data which is necessary for the application.

- \* Depends on the `.[EntityFrameworkCore](#)` project (for EF Core) since it needs to access to the migrations.
- \* Depends on the `.[Application.Contracts](#)` project to be able to access permission definitions, because the initial data seeder grants all permissions to the admin role by default.

#### #### .HttpApi Project

This project is used to define your API Controllers.

Most of the time you don't need to manually define API Controllers since ABP's [[Auto API Controllers](#)](./API/Auto-API-Controllers.md) feature creates them automagically based on your application layer. However, in case of you need to write API controllers, this is the best place to do it.

- \* Depends on the `.[Application.Contracts](#)` project to be able to inject the application service interfaces.

#### #### .HttpApi.Client Project

This is a project that defines C# client proxies to use the HTTP APIs of the solution. You can share this library to 3rd-party clients, so they can easily consume your HTTP APIs in their Dotnet applications (For other types of applications, they can still use your APIs, either manually or using a tool in their own platform)

Most of the time you don't need to manually create C# client proxies, thanks to ABP's [[Dynamic C# API Clients](#)](./API/Dynamic-CSharp-API-Clients.md) feature.

`'.HttpApi.Client.ConsoleTestApp'` project is a console application created to demonstrate the usage of the client proxies.

\* Depends on the `'.Application.Contracts'` project to be able to share the same application service interfaces and DTOs with the remote service.

> You can delete this project & dependencies if you don't need to create C# client proxies for your APIs.

#### #### .Web Project

This project contains the User Interface (UI) of the application if you are using ASP.NET Core MVC UI. It contains Razor pages, JavaScript files, CSS files, images and so on...

This project contains the main `'appsettings.json'` file that contains the connection string and other configurations of the application.

\* Depends on the `'.HttpApi'` project since the UI layer needs to use APIs and the application service interfaces of the solution.

> If you check the source code of the `'.Web.csproj'` file, you will see the references to the `'.Application'` and the `'.EntityFrameworkCore'` projects.

>

> These references are actually not needed while coding your UI layer, because the UI layer normally doesn't depend on the EF Core or the Application layer's implementation. These startup templates are ready for tiered deployment, where the API layer is hosted on a separate server than the UI layer.

>

> However, if you don't choose the `--tiered` option, these references will be in the .Web project to be able to host the Web, API and application layers in a single application endpoint.

>

> This gives you the ability to use domain entities & repositories in your presentation layer. However, this is considered as a bad practice according to DDD.

#### #### Test Projects

The solution has multiple test projects, one for each layer:

- \* `'.Domain.Tests'` is used to test the domain layer.
- \* `'.Application.Tests'` is used to test the application layer.
- \* `'.EntityFrameworkCore.Tests'` is used to test EF Core configuration and custom repositories.
- \* `'.Web.Tests'` is used to test the UI (if you are using ASP.NET Core MVC UI).
- \* `'.TestBase'` is a base (shared) project for all tests.

In addition, `'.HttpApi.Client.ConsoleTestApp'` is a console application (not an automated test project) which demonstrate the usage of HTTP APIs from a .NET application.

Test projects are prepared for integration testing;

- \* It is fully integrated into the ABP framework and all services in your application.

- \* It uses SQLite in-memory database for EF Core. For MongoDB, it uses the [\[Mongo2Go\]\(https://github.com/Mongo2Go/Mongo2Go\)](https://github.com/Mongo2Go/Mongo2Go) library.
- \* Authorization is disabled, so any application service can be easily used in tests.

You can still create unit tests for your classes which will be harder to write (because you will need to prepare mock/fake objects), but faster to run (because it only tests a single class and skips all the initialization processes).

#### #### How to Run?

Set `'.Web` as the startup project and run the application. The default username is `'admin` and the password is `'1q2w3E\*`.

See [\[Getting Started With the ASP.NET Core MVC Template\]\(../Getting-Started-AspNetCore-MVC-Template.md\)](#) for more information.

#### ### Tiered Structure

If you have selected the ASP.NET Core UI and specified the `'--tiered` option, the solution created will be a tiered solution. The purpose of the tiered structure is to be able to \*\*deploy Web applications and HTTP API to different servers\*\*:

![bookstore-visual-studio-solution-v3](../images/tiered-solution-servers.png)

- \* Browser runs your UI by executing HTML, CSS & JavaScript.
- \* Web servers host static UI files (CSS, JavaScript, image... etc.) & dynamic components (e.g. Razor pages). It performs HTTP requests to the API server to execute the business logic of the application.
- \* The API Server hosts the HTTP APIs which then use the application & domain layers of the application to perform the business logic.
- \* Finally, database server hosts your database.

So, the resulting solution allows a 4-tiered deployment, by comparing to 3-tiered deployment of the default structure explained before.

> Unless you actually need such a 4-tiered deployment, it's suggested to go with the default structure which is simpler to develop, deploy and maintain.

The solution structure is shown below:

![bookstore-rider-solution-v6](../images/bookstore-rider-solution-tiered.png)

As different from the default structure, two new projects come into play: `'.AuthServer` & `'.HttpApi.Host`.

#### #### .AuthServer Project

This project is used as an authentication server for other projects. `'.Web` project uses OpenId Connect Authentication to get identity and access tokens for the current user from the AuthServer. Then uses the access token to call the HTTP API server. HTTP API server uses bearer token authentication to obtain claims from the access token to authorize the current user.

![tiered-solution-applications](../images/tiered-solution-applications-authserver.png)

ABP uses the [OpenIddict Module](./Modules/OpenIddict.md) that uses the open-source [OpenIddict-core](<https://github.com/openiddict/openiddict-core>) library for the authentication between applications. See [OpenIddict documentation](<https://documentation.openiddict.com/>) for details about the OpenIddict and OpenID Connect protocol.

It has its own `appsettings.json` that contains database connection and other configurations.

#### #### .HttpApi.Host Project

This project is an application that hosts the API of the solution. It has its own `appsettings.json` that contains database connection and other configurations.

#### #### .Web Project

Just like the default structure, this project contains the User Interface (UI) of the application. It contains razor pages, JavaScript files, style files, images and so on...

This project contains an `appsettings.json` file, but this time it does not have a connection string because it never connects to the database. Instead, it mainly contains the endpoint of the remote API server and the authentication server.

#### #### Pre-requirements

- \* [Redis](<https://redis.io/>): The applications use Redis as a distributed cache. So, you need to have Redis installed & running.

#### #### How to Run?

You should run the application with the given order:

- \* First, run the `.AuthServer` since other applications depend on it.
- \* Then run the `.HttpApi.Host` since it is used by the `.Web` application.
- \* Finally, you can run the `.Web` project and login to the application (using `admin` as the username and `1q2w3E\*` as the password).

#### ## Blazor UI

If you choose `Blazor` as the UI Framework (using the `-u blazor` or `-u blazor-server` option), the solution will have a project named `.Blazor`. This project contains the Blazor UI application. According to your choice, it will be a Blazor WebAssembly or Blazor Server application. If Blazor WebAssembly is selected, the solution will also have a `.HttpApi.Host`. This project is an ASP.NET Core application that hosts the backend application for the Blazor single page application.

#### #### .Blazor Project (Server)

The Blazor Server project is similar to the ASP.NET Core MVC project. It replaces `.Web` project with `.Blazor` in the solution structure above. It has the same folder structure and the same application flow. Since it's an ASP.NET Core application, it can contain \*\*.cshtml\*\* files and \*\*.razor\*\* components at the same time. If routing matches a razor component, the Blazor UI will be used. Otherwise, the request will be handled by the MVC framework.

![abp solution structure blazor server](./images/layered-project-dependencies-blazor-server.png)

#### #### .Blazor Project (WebAssembly)

The Blazor WebAssembly project is a single page application that runs on the browser. You'll see it as `Blazor` project in the solution. It uses the `HttpApi.Host` project to communicate with the backend. It can't be used without the backend application. It contains only \*\*.razor\*\* components. It's a pure client-side application. It doesn't have any server-side code. Everything in this layer will be for the client side.

![abp solution structure blazor wasm](./images/layered-project-dependencies-blazor-wasm.png)

#### ### Angular UI

If you choose `Angular` as the UI framework (using the `--u angular` option), the solution is being separated into two folders:

- \* `angular` folder contains the Angular UI application, the client-side code.
- \* `aspnet-core` folder contains the ASP.NET Core solution, the server-side code.

The server-side is similar to the solution described above. `\*.HttpApi.Host` project serves the API, so the `Angular` application consumes it.

Angular application folder structure looks like below:

![angular-folder-structure](./images/angular-folder-structure.png)

Each of ABP Commercial modules is an NPM package. Some ABP modules are added as a dependency in `package.json`. These modules install with their dependencies. To see all ABP packages, you can run the following command in the `angular` folder:

```
```bash
yarn list --pattern abp
````
```

Angular application module structure:

![Angular template structure diagram](./images/angular-template-structure-diagram.png)

#### #### AppModule

`AppModule` is the root module of the application. Some of the ABP modules and some essential modules are imported to `AppModule`.

ABP Config modules have also been imported to `AppModule` for initial requirements of the lazy-loadable ABP modules.

#### #### AppRoutingModule

There are lazy-loadable ABP modules in the `AppRoutingModule` as routes.

> Paths of ABP Modules should not be changed.

You should add `'routes'` property in the `'data'` object to add a link on the menu to redirect to your custom pages.

```
```js
{
  path: 'dashboard',
  loadChildren: () => import('./dashboard/dashboard.module').then(m =>
m.DashboardModule),
  canActivate: [AuthGuard, PermissionGuard],
  data: {
    routes: {
      name: 'ProjectName::Menu:Dashboard',
      order: 2,
      iconClass: 'fa fa-dashboard',
      requiredPolicy: 'ProjectName.Dashboard.Host'
    } as ABP.Route
  }
}..
```

```

In the above example;

- \* If the user is not logged in, AuthGuard blocks access and redirects to the login page.
- \* PermissionGuard checks the user's permission with the `'requiredPolicy'` property of the `'routes'` object. If the user is not authorized to access the page, the 403 page appears.
- \* The `'name'` property of `'routes'` is the menu link label. A localization key can be defined.
- \* The `'iconClass'` property of the `'routes'` object is the menu link icon class.
- \* The `'requiredPolicy'` property of the `'routes'` object is the required policy key to access the page.

After the above `'routes'` definition, if the user is authorized, the dashboard link will appear on the menu.

#### #### Shared Module

The modules that may be required for all modules have been imported to the `' SharedModule'`. You should import `' SharedModule'` to all modules.

See the [\[Sharing Modules\]](#)(<https://angular.io/guide/sharing-ngmodules>) document.

#### #### Environments

The files under the `'src/environments'` folder have the essential configuration of the application.

#### #### Home Module

Home module is an example lazy-loadable module that loads on the root address of the application.

#### #### Styles

The required style files are added to the `styles` array in `angular.json`. `AppComponent` loads some style files lazily via `LazyLoadService` after the main bundle is loaded to shorten the first rendering time.

#### #### Testing

You should create your tests in the same folder as the file you want to test.

See the [testing document](<https://angular.io/guide/testing>).

#### #### Depended Packages

- \* [NG Bootstrap](<https://ng-bootstrap.github.io/>) is used as UI component library.
- \* [NGXS](<https://www.ngxs.io/>) is used as state management library.
- \* [angular-oauth2-oidc](<https://github.com/manfredsteyer/angular-oauth2-oidc>) is used to support for OAuth 2 and OpenId Connect (OIDC).
- \* [Chart.js](<https://www.chartjs.org/>) is used to create widgets.
- \* [ngx-validate](<https://github.com/ng-turkey/ngx-validate>) is used for dynamic validation of reactive forms.

#### ### React Native

If the `--m react-native` option is specified in the new project command, the solution includes the [React Native](<https://reactnative.dev/>) application in the `react-native` folder.

The server-side is similar to the solution described above. `\*.HttpApi.Host` project serves the API, so the React Native application consumes it.

The React Native application was generated with [Expo](<https://expo.io/>). Expo is a set of tools built around React Native to help you quickly start an app and, while it has many features.

React Native application folder structure as like below:

- ```
![react-native-folder-structure](../images/react-native-folder-structure.png)
```
- * `App.js` is the bootstrap component of the application.
 - * `Environment.js` file has the essential configuration of the application. `prod` and `dev` configurations are defined in this file.
 - * [Contexts](<https://reactjs.org/docs/context.html>) are created in the `src/context` folder.
 - * [Higher order components](<https://reactjs.org/docs/higher-order-components.html>) are created in the `src/hocs` folder.
 - * [Custom hooks](<https://reactjs.org/docs/hooks-custom.html#extracting-a-custom-hook>) are created in `src/hooks`.
 - * [Axios interceptors](<https://github.com/axios/axios#interceptors>) are created in the `src/interceptors` folder.
 - * Utility functions are exported from `src/utils` folder.

Components

Components that can be used on all screens are created in the `src/components` folder. All components have been created as a function that is able to use [hooks](<https://reactjs.org/docs/hooks-intro.html>).

Screens

![react-native-navigation-structure](../images/react-native-navigation-structure.png)

Screens are created by creating folders that separate their names in the `src/screens` folder. Certain parts of some screens can be split into components.

Each screen is used in a navigator in the `src/navigators` folder.

Navigation

[React Navigation](https://reactnavigation.org/) is used as a navigation library. Navigators are created in the `src/navigators`. A [drawer](https://reactnavigation.org/docs/drawer-based-navigation/) navigator and several [stack](https://reactnavigation.org/docs/hello-react-navigation/#installing-the-stack-navigator-library) navigators have been created in this folder. See the [above diagram](#screens) for the navigation structure.

State Management

[Redux](https://redux.js.org/) is used as a state management library. [Redux Toolkit](https://redux-toolkit.js.org/) library is used as a toolset for efficient Redux development.

Actions, reducers, sagas and selectors are created in the `src/store` folder. Store folder is as below:

![react-native-store-folder](../images/react-native-store-folder.png)

- * [**Store**](https://redux.js.org/basics/store) is defined in the `src/store/index.js` file.
- * [**Actions**](https://redux.js.org/basics/actions/) are payloads of information that send data from your application to your store.
- * [**Reducers**](https://redux.js.org/basics/reducers) specify how the application's state changes in response to actions sent to the store.
- * [**Redux-Saga**](https://redux-saga.js.org/) is a library that aims to make application side effects (i.e. asynchronous things like data fetching and impure things like accessing the browser cache) easier to manage. Sagas are created in the `src/store/sagas` folder.
- * [**Reselect**](https://github.com/reduxjs/reselect) library is used to create memoized selectors. Selectors are created in the `src/store/selectors` folder.

APIs

[Axios](https://github.com/axios/axios) is used as an HTTP client library. An Axios instance has exported from `src/api/API.js` file to make HTTP calls with the same config. `src/api` folder also has the API files that have been created for API calls.

Theming

[Native Base](https://nativebase.io/) is used as UI components library. Native Base components can customize easily. See the [Native Base customize](https://docs.nativebase.io/customizing-components) documentation. We followed the same way.

```
* Native Base theme variables are in the 'src/theme/variables' folder.  
* Native Base component styles are in the 'src/theme/components' folder.  
These files have been generated with Native Base's 'ejectTheme' script.  
* Styles of components override with the files under the  
'src/theme/overrides' folder.
```

Testing

Unit tests will be created.

See the [[Testing Overview](#)](<https://reactjs.org/docs/testing.html>) document.

Depended Libraries

- * [[Native Base](#)](<https://nativebase.io/>) is used as UI components library.
- * [[React Navigation](#)](<https://reactnavigation.org/>) is used as navigation library.
- * [[Axios](#)](<https://github.com/axios/axios>) is used as an HTTP client library.
- * [[Redux](#)](<https://redux.js.org/>) is used as state management library.
- * [[Redux Toolkit](#)](<https://redux-toolkit.js.org/>) library is used as a toolset for efficient Redux development.
- * [[Redux-Saga](#)](<https://redux-saga.js.org/>) is used to manage asynchronous processes.
- * [[Redux Persist](#)](<https://github.com/rt2zz/redux-persist>) is used as state persistence.
- * [[Reselect](#)](<https://github.com/reduxjs/reselect>) is used to create memoized selectors.
- * [[i18n-js](#)](<https://github.com/fnando/i18n-js>) is used as i18n library.
- * [[expo-font](#)](<https://docs.expo.io/versions/latest/sdk/font/>) library allows loading fonts easily.
- * [[Formik](#)](<https://github.com/jaredpalmer/formik>) is used to build forms.
- * [[Yup](#)](<https://github.com/jquense/yup>) is used for form validations.

Social / External Logins

If you want to configure social/external logins for your application, please follow the [[Social/External Logins](#)]([..../Authentication/Social-External-Logins.md](#)) document.

What's Next?

- [[The getting started document](#)]([..../Getting-Started.md](#)) explains how to create a new application in a few minutes.
- [[The application development tutorial](#)]([..../Tutorials/Part-1.md](#)) explains step by step application development.

5.3 Application (Single Layer)

Application (Single Layer) Startup Template

Introduction

This template provides a simple solution structure with a single project. This document explains that solution structure in details.

The Difference Between the Application Startup Templates

ABP's [Application Startup Template](Application.md) provides a well-organized and layered solution to create maintainable business applications based on the [Domain Driven Design](../Domain-Driven-Design.md) (DDD) practices. However, some developers find this template a little bit complex for simple and short-term applications. The single-layer application template has been created to provide a simpler development model for such applications. This template has the same functionality, features and modules on runtime with the [Application Startup Template](Application.md) but the development model is minimal and everything is in a single project (`.csproj`).

How to Start with It?

You can use the [ABP CLI](../CLI.md) to create a new project using this startup template. Alternatively, you can generate a CLI command for this startup template from the [Get Started](https://abp.io/get-started) page. In this section, we will use the ABP CLI.

Firstly, install the ABP CLI if you haven't installed it before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
````
```

Then, use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.BookStore -t app-nolayers
````
```

- * `Acme.BookStore` is the solution name, like *YourCompany.YourProduct*. You can use single-level, two-level or three-level naming.
- * In this example, the `-t` (or `--template`) option specifies the template name.

Specify the UI Framework

This template provides multiple UI frameworks:

- * `mvc`: ASP.NET Core MVC UI with Razor Pages (default)
- * `blazor`: Blazor UI
- * `blazor-server`: Blazor Server UI
- * `angular`: Angular UI
- * `none`: Without UI (for HTTP API development)

Use the `-u` (or `--ui`) option to specify the UI framework while creating the solution:

```
```bash
abp new Acme.BookStore -t app-nolayers -u angular
````
```

This example specifies the UI type (the `-u` option) as `angular`. You can also specify `mvc`, `blazor`, `blazor-server` or `none` for the UI type.

Specify the Database Provider

This template supports the following database providers:

- `ef`: Entity Framework Core (default)
- `mongodb`: MongoDB

Use the `-d` (or `--database-provider`) option to specify the database provider while creating the solution:

```
```bash
abp new Acme.BookStore -t app-nolayers -d mongodb
```
```

Solution Structure

If you don't specify any additional options while creating an `app-nolayers` template, you will have a solution as shown below:

```

```

In the next sections, we will explain the structure based on this example. Your startup solution can be slightly different based on your preferences.

Folder Structure

Since this template provides a single-project solution, we've separated concerns into folders instead of projects. You can see the pre-defined folders as shown below:

```

```

- * Define your database mappings (for [EF Core](./Entity-Framework-Core.md) or [MongoDB](./MongoDB.md)) and [repositories](./Repositories.md) in the `'Data'` folder.
- * Define your `[entities]`(./Entities.md) in the `'Entities'` folder.
- * Define your UI localization keys/values in the `'Localization'` folder.
- * Define your UI menu items in the `'Menus'` folder.
- * Define your `[object-to-object mapping]`(./Object-To-Object-Mapping.md) classes in the `'ObjectMapping'` folder.
- * Define your UI pages (Razor Pages) in the `'Pages'` folder (create `'Controllers'` and `'Views'` folder yourself if you prefer the MVC pattern).
- * Define your `[application services]`(./Application-Services.md) in the `'Services'` folder.

How to Run?

Before running the application, you need to create the database and seed the initial data. To do that, you can run the following command in the directory of your project (in the same folder of the `'.csproj'` file):

```
```bash
dotnet run --migrate-database
```
```

This command will create the database and seed the initial data for you. Then you can run the application with any IDE that supports .NET or by running the `dotnet run` command in the directory of your project. The default username is `'admin'` and the password is `'1q2w3E*'`.

> While creating a database & applying migrations seem only necessary for relational databases, you should run this command even if you choose a NoSQL database provider (like MongoDB). In that case, it still seeds the initial data which is necessary for the application.

The Angular UI

If you choose `Angular` as the UI framework, the solution will be separated into two folders:

- * An `angular` folder that contains the Angular UI application, the client-side code.
- * An `aspnet-core` folder that contains the ASP.NET Core solution (a single project), the server-side code.

The server-side is similar to the solution described in the **Solution Structure** section above. This project serves the API, so the Angular application can consume it.

The client-side application consumes the HTTP APIs as mentioned. You can see the folder structure of the Angular project shown below:

5.4 Module

Module Startup Template

This template can be used to create a **reusable [application module](./Modules/Index.md)** based on the [module development best practices & conventions](./Best-Practices/Index.md). It is also suitable for creating **microservices** (with or without UI).

How to Start With?

You can use the [ABP CLI](./CLI.md) to create a new project using this startup template. Alternatively, you can generate a CLI command from the [Get Started](https://abp.io/get-started) page. CLI approach is used here.

First, install the ABP CLI if you haven't installed before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
```
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.IssueManagement -t module
```
```

- `Acme.IssueManagement` is the solution name, like **YourCompany.YourProduct**. You can use single level, two-levels or three-levels naming.

Without User Interface

The template comes with MVC, Blazor & Angular user interfaces by default. You can use `--no-ui` option to not include any of these UI layers.

```
```bash
abp new Acme.IssueManagement -t module --no-ui
````
```

Solution Structure

Based on the options you've specified, you will get a slightly different solution structure. If you don't specify any option, you will have a solution like shown below:

![issuemangement-module-solution](../images/issuemangement-module-solution.png)

Projects are organized as 'src', 'test' and 'host' folders:

- * 'src' folder contains the actual module which is layered based on [DDD](../Domain-Driven-Design.md) principles.
- * 'test' folder contains unit & integration tests.
- * 'host' folder contains applications with different configurations to demonstrate how to host the module in an application. These are not a part of the module, but useful on development.

The diagram below shows the layers & project dependencies of the module:

![layered-project-dependencies-module](../images/layered-project-dependencies-module.png)

Each section below will explain the related project & its dependencies.

.Domain.Shared Project

This project contains constants, enums and other objects these are actually a part of the domain layer, but needed to be used by all layers/projects in the solution.

An 'IssueType' enum and an 'IssueConsts' class (which may have some constant fields for the 'Issue' entity, like 'MaxTitleLength') are good candidates for this project.

- This project has no dependency to other projects in the solution. All other projects depend on this directly or indirectly.

.Domain Project

This is the domain layer of the solution. It mainly contains [entities, aggregate roots](../Entities.md), [domain services](../Domain-Services.md), value types, [repository interfaces](../Repositories.md) and other domain objects.

An 'Issue' entity, an 'IssueManager' domain service and an 'IIssueRepository' interface are good candidates for this project.

- Depends on the `Domain.Shared` because it uses constants, enums and other objects defined in that project.

.Application.Contracts Project

This project mainly contains [application service](../Application-Services.md) **interfaces** and [Data Transfer Objects](../Data-Transfer-Objects.md) (DTO) of the application layer. It does exists to separate interface & implementation of the application layer. In this way, the interface project can be shared to the clients as a contract package.

An `IIssueAppService` interface and an `IssueCreationDto` class are good candidates for this project.

- Depends on the `Domain.Shared` because it may use constants, enums and other shared objects of this project in the application service interfaces and DTOs.

.Application Project

This project contains the [application service](../Application-Services.md) **implementations** of the interfaces defined in the `Application.Contracts` project.

An `IssueAppService` class is a good candidate for this project.

- Depends on the `Application.Contracts` project to be able to implement the interfaces and use the DTOs.
- Depends on the `Domain` project to be able to use domain objects (entities, repository interfaces... etc.) to perform the application logic.

.EntityFrameworkCore Project

This is the integration project for EF Core. It defines the `DbContext` and implements repository interfaces defined in the `Domain` project.

- Depends on the `Domain` project to be able to reference to entities and repository interfaces.
- > You can delete this project if you don't want to support EF Core for your module.

.MongoDB Project

This is the integration project for MongoDB.

- Depends on the `Domain` project to be able to reference to entities and repository interfaces.
- > You can delete this project if you don't want to support MongoDB for your module.

Test Projects

The solution has multiple test projects, one for each layer:

- `Domain.Tests` is used to test the domain layer.
- `Application.Tests` is used to test the application layer.

- `'.EntityFrameworkCore.Tests'` is used to test EF Core configuration and custom repositories.
- `'.MongoDB.Tests'` is used to test MongoDB configuration and custom repositories.
- `'.TestBase'` is a base (shared) project for all tests.

In addition, `'.HttpApi.Client.ConsoleTestApp'` is a console application (not an automated test project) which demonstrate the usage of HTTP APIs from a Dotnet application.

Test projects are prepared for integration testing;

- It is fully integrated to ABP framework and all services in your application.
- It uses SQLite in-memory database for EF Core. For MongoDB, it uses the [Mongo2Go](<https://github.com/Mongo2Go/Mongo2Go>) library.
- Authorization is disabled, so any application service can be easily used in tests.

You can still create unit tests for your classes which will be harder to write (because you will need to prepare mock/fake objects), but faster to run (because it only tests a single class and skips all initialization process).

> Domain & Application tests are using EF Core. If you remove EF Core integration or you want to use MongoDB for testing these layers, you should manually change project references & module dependencies.

Host Projects

The solution has a few host applications to run your module. Host applications are used to run your module in a fully configured application. It is useful on development. Host applications includes some other modules in addition to the module being developed:

Host applications support two types of scenarios.

Single (Unified) Application Scenario

If your module has a UI, then `'.Web.Unified'` application is used to host the UI and API on a single point. It has its own `appsettings.json` file (that includes the database connection string) and EF Core database migrations.

For the `'.Web.Unified'` application, there is a single database, named `YourProjectName_Unified` (like *IssueManagement_Unified* for this sample).

> If you've selected the `--no-ui` option, this project will not be in your solution.

How to Run?

Set `host/YourProjectName.Web.Unified` as the startup project, run `Update-Database` command for the EF Core from Package Manager Console and run your application. Default username is `admin` and password is `1q2w3E*`.

Separated Deployment & Databases Scenario

In this scenario, there are three applications;

- * `'.AuthServer'` application is an authentication server used by other applications. It has its own `appsettings.json` that contains database connection and other configurations.
- * `'.HttpApi.Host'` hosts the HTTP API of the module. It has its own `appsettings.json` that contains database connections and other configurations.
- * `'.Web.Host'` host the UI of the module. This project contains an `appsettings.json` file, but it does not have a connection string because it never connects to the database. Instead, it mainly contains endpoint of the remote API server and the authentication server.

The diagram below shows the relation of the applications:

![tiered-solution-applications](./images/tiered-solution-applications.png)

`'.Web.Host'` project uses OpenId Connect Authentication to get identity and access tokens for the current user from the `'.AuthServer'`. Then uses the access token to call the `'.HttpApi.Host'`. HTTP API server uses bearer token authentication to obtain claims from the access token to authorize the current user.

Pre-requirements

- * [Redis](https://redis.io/): The applications use Redis as a distributed cache. So, you need to have Redis installed & running.

How to Run?

You should run the application with the given order:

- First, run the `'.AuthServer'` since other applications depends on it.
- Then run the `'.HttpApi.Host'` since it is used by the `'.Web.Host'` application.
- Finally, you can run the `'.Web.Host'` project and login to the application using `admin` as the username and `1q2w3E*` as the password.

UI

Angular UI

The solution will have a folder called `angular` in it. This is where the Angular client-side code is located. When you open that folder in an IDE, the folder structure will look like below:

![Folder structure of ABP Angular module project](./images/angular-module-folder-structure.png)

- * `_angular/projects/issue-management_` folder contains the Angular module project.
- * `_angular/projects/dev-app_` folder contains a development application that runs your module.

The server-side is similar to the solution described above. `*.HttpApi.Host` project serves the API and the `Angular` demo application consumes it. You will not need to run the `'.Web.Host'` project though.

How to Run the Angular Development App

For module development, you will need the `dev-app` project up and running. So, here is how we can start the development server.

First, we need to install dependencies:

1. Open your terminal at the root folder, i.e. `angular`.
2. Run `yarn` or `npm install`.

The dependencies will be installed and some of them are ABP modules published as NPM packages. To see all ABP packages, you can run the following command in the `angular` folder:

```
```bash
yarn list --pattern abp
````
```

> There is no equivalent of this command in npm.

The module you will develop depends on two of these ABP packages: `_@abp/ng.core_` and `_@abp/ng.theme.shared_`. Rest of the ABP modules are included in `_package.json_` because of the `dev-app` project.

Once all dependencies are installed, follow the steps below to serve your development app:

1. Make sure `AuthServer` and `*HttpApi.Host` projects are up and running.
2. Open your terminal at the root folder, i.e. `angular`.
3. Run `yarn start` or `npm start`.

![ABP Angular module dev-app project](../images/angular-module-dev-app-project.png)

The issue management page is empty in the beginning. You may change the content in `'IssueManagementComponent'` at the `_angular/projects/issue-management/src/lib/issue-management.component.ts_` path and observe that the view changes accordingly.

Now, let's have a closer look at some key elements of your project.

Main Module

`'IssueManagementModule'` at the `_angular/projects/issue-management/src/lib/issue-management.module.ts_` path is the main module of your module project. There are a few things worth mentioning in it:

- Essential ABP modules, i.e. `'CoreModule'` and `'ThemeSharedModule'`, are imported.
- `'IssueManagementRoutingModule'` is imported.
- `'IssueManagementComponent'` is declared.
- It is prepared for configurability. The `'forLazy'` static method enables [a configuration to be passed to the module when it is loaded by the router](<https://volosoft.com/blog/how-to-configure-angular-modules-loaded-by-the-router>).

Main Routing Module

`'IssueManagementRoutingModule'` at the `_angular/projects/issue-management/src/lib/issue-management-routing.module.ts` path is the main routing module of your module project. It currently does two things:

- Loads `'DynamicLayoutComponent'` at base path it is given.
- Loads `'IssueManagementComponent'` as child to the layout, again at the given base path.

You can rearrange this module to load more than one component at different routes, but you need to update the route provider at `_angular/projects/issue-management/config/src/providers/route.provider.ts` to match the new routing structure with the routes in the menu. Please check [\[Modifying the Menu\]\(../UI/Angular/Modifying-the-Menu.md\)](#) to see how route providers work.

Config Module

There is a config module at the `_angular/projects/issue-management/config/src/issue-management-config.module.ts` path. The static `'forRoot'` method of this module is supposed to be called at the route level. So, you may assume the following will take place:

```
'''js
@NgModule({
  imports: [
    /* other imports */

    IssueManagementConfigModule.forRoot(),
  ],
  /* rest of the module meta data */
})
export class AppModule {}'''
```

You can use this static method to configure an application that uses your module project. An example of such configuration is already implemented and the `'ISSUE_MANAGEMENT_ROUTE_PROVIDERS'` token is provided here. The method can take options which enables further configuration possibilities.

The difference between the `'forRoot'` method of the config module and the `'forLazy'` method of the main module is that, for smallest bundle size, the former should only be used when you have to configure an app before your module is even loaded.

Testing Angular UI

Please see the [\[testing document\]\(../UI/Angular/Testing.md\)](#).

5.5 Console

Console Application Startup Template

This template is used to create a minimalist console application project.

How to Start With?

First, install the [ABP CLI](./CLI.md) if you haven't installed before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
```
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.MyConsoleApp -t console
```
```

`Acme.MyConsoleApp` is the solution name, like **YourCompany.YourProduct**. You can use single level, two-levels or three-levels naming.

Solution Structure

After you use the above command to create a solution, you will have a solution like shown below:

![basic-console-application-solution](./images/basic-console-application-solution.png)

* `HelloWorldService` is a sample service that implements the `ITransientDependency` interface to register this service to the [dependency injection](./Dependency-Injection.md) system.

5.6 WPF

WPF Application Startup Template

This template is used to create a minimalist WPF application project.

How to Start With?

First, install the [ABP CLI](./CLI.md) if you haven't installed before:

```
```bash
dotnet tool install -g Volo.Abp.Cli
```
```

Then use the `abp new` command in an empty folder to create a new solution:

```
```bash
abp new Acme.MyWpfApp -t wpf
```
```

`Acme.MyWpfApp` is the solution name, like **YourCompany.YourProduct**. You can use single level, two-levels or three-levels naming.

Solution Structure

After you use the above command to create a solution, you will have a solution like shown below:

![basic-wpf-application-solution](./images/basic-wpf-application-solution.png)

* `HelloWorldService` is a sample service that implements the `ITransientDependency` interface to register this service to the [dependency injection](./Dependency-Injection.md) system.

6 Fundamentals

6.1 Application Startup

ABP Application Startup

You typically use the [ABP CLI](CLI.md)'s `abp new` command to [get started](Getting-Started.md) with one of the pre-built [startup solution templates](Startup-Templates/Index.md). When you do that, you generally don't need to know the details of how the ABP Framework is integrated with your application or how it is configured and initialized. The startup template also comes with the fundamental ABP packages and [application modules](Modules/Index) are pre-installed and configured for you.

> It is always suggested to [get started with a startup template](Getting-Started.md) and modify it for your requirements. Read this document only if you want to understand the details or if you need to modify how the ABP Framework starts.

While the ABP Framework has a lot of features and integrations, it is built as a lightweight and modular framework. It consists of [hundreds of NuGet and NPM packages](https://abp.io/packages), so you can only use the features you need. If you follow the [Getting Started with an Empty ASP.NET Core MVC / Razor Pages Application](Getting-Started-AspNetCore-Application.md) document, you'll see how easy it is to install the ABP Framework into an empty ASP.NET Core project from scratch. You only need to install a single NuGet package and make a few small changes.

This document is for who wants to better understand how the ABP Framework is initialized and configured on startup.

Installing to a Console Application

A .NET Console application is the minimalist .NET application. So, it is best to show the installing of the ABP Framework to a console application as a minimalist example.

If you [create a new console application with Visual Studio](https://learn.microsoft.com/en-us/dotnet/core/tutorials/with-visual-studio) (for .NET 7.0 or later), you will see the following solution structure (I named the solution as `MyConsoleDemo`):

![app-startup-console-initial](images/app-startup-console-initial.png)

This example uses the [top level statements](https://learn.microsoft.com/en-us/dotnet/csharp/whats-new/tutorials/top-level-statements), so it consists of only a single line of code.

The first step is to install the [Volo.Abp.Core](https://www.nuget.org/packages/Volo.Abp.Core) NuGet package, which is the most core NuGet package of the ABP framework. You can install it

using the ABP CLI. Execute the following command in the folder of the .csproj file that you want to install the package on:

```
```bash
abp add-package Volo.Abp.Core
````
```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.Core).

Alternatively, you can use a command-line terminal in the root folder of the project (the folder containing the 'MyConsoleDemo.csproj' file, for this example):

```
```bash
dotnet add package Volo.Abp.Core
````
```

After adding the NuGet package, we should create a root [module class](Module-Development-Basics.md) for our application. We can create the following class in the project:

```
```csharp
using Volo.Abp.Modularity;

namespace MyConsoleDemo
{
 public class MyConsoleDemoModule : AbpModule
 {
 }
}
````
```

This is an empty class deriving from the 'AbpModule' class. It is the main class that you will control your application's dependencies with, and implement your configuration and startup/shutdown logic. For more information, please check the [Modularity](Module-Development-Basics.md) document.

As the second and the last step, change the 'Program.cs' as shown in the following code block:

```
```csharp
using MyConsoleDemo;
using Volo.Abp;

// 1: Create the ABP application container
using var application = await
AbpApplicationFactory.CreateAsync<MyConsoleDemoModule>();

// 2: Initialize/start the ABP Framework (and all the modules)
await application.InitializeAsync();

Console.WriteLine("ABP Framework has been started...");

// 3: Stop the ABP Framework (and all the modules)
await application.ShutdownAsync();
````
```

....

That's all. Now, ABP Framework is installed, integrated, started and stopped in your application. From now, you can install [ABP packages](<https://abp.io/packages>) to your application whenever you need them.

Installing a Framework Package

If you want to send emails from your .NET application, you can use .NET's standard [SmtpClient class](<https://learn.microsoft.com/en-us/dotnet/api/system.net.mail.smtpclient>). ABP also provides an `IEmailSender` service that simplifies [sending emails](Emailling.md) and configuring the email settings in a central place. If you want to use it, you should install the [Volo.Abp.Emailing](<https://www.nuget.org/packages/Volo.Abp.Emailing>) NuGet package to your project:

```
```bash
dotnet add package Volo.Abp.Emailing
```
```

Once you add a new ABP package/module, you also need to specify the module dependency from your module class. So, change the `MyConsoleDemoModule` class as shown below:

```
```csharp
using Volo.Abp.Emailing;
using Volo.Abp.Modularity;

namespace MyConsoleDemo
{
 [DependsOn(typeof(AbpEmailingModule))] // Added the module dependency
 public class MyConsoleDemoModule : AbpModule
 {
 }
}
````
```

I've just added a `[DependsOn]` attribute to declare that I want to use the ABP Emailing Module (`AbpEmailingModule`). Now, I can use the `IEmailSender` service in my `Program.cs`:

```
```csharp
using Microsoft.Extensions.DependencyInjection;
using MyConsoleDemo;
using Volo.Abp;
using Volo.Abp.Emailing;

using var application = await
AbpApplicationFactory.CreateAsync<MyConsoleDemoModule>();
await application.InitializeAsync();

// Sending emails using the IEmailSender service
var emailsender =
application.ServiceProvider.GetRequiredService<IEmailSender>();
await emailsender.SendAsync(
 to: "info@acme.com",
 subject: "Hello from ABP!",
 body: "This is a test email sent via ABP Emailing module.");
````
```

```
        subject: "Hello World",
        body: "My message body..."
    );

await application.ShutdownAsync();
````
```

> If you run that application, you get a runtime error indicating that the email sending settings haven't been done yet. You can check the [\[Email Sending document\]](#)(Emailing.md) to learn how to configure it.

That's all. Install an ABP NuGet package, add the module dependency (using the `[DependsOn]` attribute) and use any service inside the NuGet package.

The [\[ABP CLI\]](#)(CLI.md) already has a special command to perform the addition of an ABP NuGet and also adding the `[DependsOn]` attribute to your module class for you with a single command:

```
```bash
abp add-package Volo.Abp.Emailing
````
```

We suggest you to use the `'abp add-package'` command instead of manually doing it.

#### ## `AbpApplicationFactory`

`'AbpApplicationFactory'` is the main class that creates an ABP application container. It provides a single static `'CreateAsync'` (and `'Create'` if you can't use asynchronous programming) method with multiple overloads. Let's investigate these overloads to understand where you can use them.

The first overload gets a generic module class parameter as we've used before in this document:

```
```csharp
AbpApplicationFactory.CreateAsync<MyConsoleDemoModule>();
````
```

The generic class parameter should be the root module class of your application. All the other modules are resolved as dependencies of that module.

The second overload gets the module class as a `'Type'` parameter, instead of the generic parameter. So, the previous code block could be re-written as shown below:

```
```csharp
AbpApplicationFactory.CreateAsync(typeof(MyConsoleDemoModule));
````
```

Both overloads work exactly the same. So, you can use the second one if you don't know the module class type on development time and you (somehow) calculate it on runtime.

If you create one of the methods above, ABP creates an internal service collection (`'IServiceCollection'`) and an internal service provider (`'IServiceProvider'`) to setup the [\[dependency injection\]](#)(Dependency-

`Injection.md`) system internally. Notice that we've used the `'application.ServiceProvider'` property in the *\*Installing a Framework Package\** section to resolve the `'IEmailSender'` service from the dependency injection system.

The next overload gets an `'IServiceCollection'` parameter from you to allow you to setup the dependency injection system yourself, or integrate to another framework (like ASP.NET Core) that also sets up the dependency injection system internally.

We can change the `'Program.cs'` as shown below to externally manage the dependency injection setup:

```
```csharp
using Microsoft.Extensions.DependencyInjection;
using MyConsoleDemo;
using Volo.Abp;

// 1: Manually created the IServiceCollection
IServiceCollection services = new ServiceCollection();

// 2: Pass the IServiceCollection externally to the ABP Framework
using var application = await AbpApplicationFactory
    .CreateAsync<MyConsoleDemoModule>(services);

// 3: Manually built the IServiceProvider object
IServiceProvider serviceProvider = services.BuildServiceProvider();

// 4: Pass the IServiceProvider externally to the ABP Framework
await application.InitializeAsync(serviceProvider);

Console.WriteLine("ABP Framework has been started...");

await applicationShutdown();
```

```

In this example, we've used .NET's standard dependency injection container. The `'services.BuildServiceProvider()'` call creates the standard container. However, ABP provides an alternative extension method, `'BuildServiceProviderFromFactory()'`, that properly works even if you are using another dependency injection container:

```
```csharp
IServiceProvider serviceProvider =
services.BuildServiceProviderFromFactory();

> You can check the [Autofac Integration](Autofac-Integration.md) document if
you want to learn how you can integrate the [Autofac](https://autofac.org/)
dependency injection container with the ABP Framework.

```

Finally, the `'CreateAsync'` method has a last overload that takes the module class name as a `'Type'` parameter and a `'IServiceCollection'` object. So, we could re-write the last `'CreateAsync'` method usage as in the following code block:

```
```csharp
using var application = await AbpApplicationFactory

```

```
 .CreateAsync(typeof(MyConsoleDemoModule), services);
}
```

> All of the `CreateAsync` method overloads have `Create` counterparts. If your application type can not utilize asynchronous programming (that means you can't use the `await` keyword), then you can use the `Create` method instead of the `CreateAsync` method.

#### ### AbpApplicationCreationOptions

All of the `CreateAsync` overloads can get an optional `Action<AbpApplicationCreationOptions>` parameter to configure the options that are used on the application creation. See the following example:

```
```csharp  
using var application = await AbpApplicationFactory  
    .CreateAsync<MyConsoleDemoModule>(options =>  
    {  
        options.ApplicationName = "MyApp";  
    });  
```
```

We've passed a lambda method to configure the `ApplicationName` option. Here's a list of all standard options:

- \* `ApplicationName`: A human-readable name for the application. It is a unique value for an application.
- \* `Configuration`: Can be used to setup the [application configuration](Configuration.md) when it is not provided by the hosting system. It is not needed for ASP.NET Core and other .NET hosted applications. However, if you've used `AbpApplicationFactory` with an internal service provider, you can use this option to configure how the application configuration is built.
- \* `Environment`: Environment name for the application.
- \* `PlugInSources`: A list of plugin sources. See the [Plug-In Modules documentation](PlugIn-MODULES) to learn how to work with plugins.
- \* `Services`: The `IServiceCollection` object that can be used to register service dependencies. You generally don't need that, because you configure your services in your [module class](Module-Development-Basics.md). However, it can be used while writing extension methods for the `AbpApplicationCreationOptions` class.

#### #### The ApplicationName option

As defined above, the `ApplicationName` option is a human-readable name for the application. It is a unique value for an application.

`ApplicationName` is used by the ABP Framework in several places to distinguish the application. For example, the [audit logging](Audit-Logging.md) system saves the `ApplicationName` in each audit log record written by the related application, so you can understand which application has created the audit log entry. So, if your system consists of multiple applications (like a microservice solution) that are saving audit logs to a single point, you should be sure that each application has a different `ApplicationName`.

The `ApplicationName` property's value is set automatically from the \*\*entry assembly's name\*\* (generally, the project name in a .NET solution) by

default, which is proper for most cases, since each application typically has a unique entry assembly name.

There are two ways to set the application name to a different value. In this first approach, you can set the `ApplicationName` property in your application's [configuration](Configuration.md). The easiest way is to add an `ApplicationName` field to your `appsettings.json` file:

```
```json
{
    "ApplicationName": "Services.Ordering"
}...```

```

Alternatively, you can set `AbpApplicationCreationOptions.ApplicationName` while creating the ABP application. You can find the `AddApplication` or `AddApplicationAsync` call in your solution (typically in the `Program.cs` file), and set the `ApplicationName` option as shown below:

```
```csharp
await builder.AddApplicationAsync<OrderingServiceHttpApiHostModule>(options
=>
{
 options.ApplicationName = "Services.Ordering";
});```

```

#### #### IApplicationInfoAccessor

If you need to access the `ApplicationName` later in your solution, you can inject the `IApplicationInfoAccessor` service and get the value from its `ApplicationName` property.

`IApplicationInfoAccessor` also provides an `InstanceId` value, that is a random GUID value that is generated when your application starts. You can use that value to distinguish application instances from each other.

#### ## IAbpApplication

`AbpApplicationFactory` returns an `IAbpApplication` object from its `CreateAsync` (or `Create`) method. `IAbpApplication` is the main container for an ABP application. It is also registered to the [dependency injection](Dependency-Injection.md) system, so you can inject `IAbpApplication` in your services to use its properties and methods.

Here's a list of `IAbpApplication` properties you may want to know:

- \* `StartupModuleType`: Gets the root module of the application that was used while creating the application container (on the `AbpApplicationFactory.CreateAsync` method).
- \* `Services`: A list of all service registrations (the `IServiceCollection` object). You can not add new services to this collection after application initialization (you can actually add, but it won't have any effect).
- \* `ServiceProvider`: A reference to the root service provider used by the application. This can not be used before initializing the application. If you need to resolve non-singleton services from that `IServiceProvider` object, always create a new service scope and dispose it after usage. Otherwise, your application will have memory leak problems. See the *\*Releasing/Disposing*

*Services\** section of the [dependency injection](#) (Dependency-Injection.md) document for more information about service scopes.

\* **'Modules'**: A read-only list of all the modules loaded into the current application. Alternatively, you can inject the **'IModuleContainer'** service if you need to access the module list in your application code.

The `'IAbpApplication'` interface extends the `'IApplicationInfoAccessor'` interface, so you can get the `'ApplicationName'` and `'InstanceId'` values from it. However, if you only need to access these properties, inject and use the `'IApplicationInfoAccessor'` service instead.

`'IAbpApplication'` is disposable. Always dispose of it before exiting your application.

## ## IAbpHostEnvironment

Sometimes, while creating an application, we need to get the current hosting environment and take actions according to that. In such cases, we can use some services such as [[IWebHostEnvironment](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.iwebhostenvironment?view=aspnetcore-7.0)] (<https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.iwebhostenvironment?view=aspnetcore-7.0>) or [[IWebAssemblyHostEnvironment](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.webassembly.hosting.iwebassemblyhostenvironment)] (<https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.components.webassembly.hosting.iwebassemblyhostenvironment>) provided by .NET, in the final application.

However, we can not use these services in a class library, which is used by the final application. ABP Framework provides the '[IAbpHostEnvironment](#)' service, which allows you to get the current environment name whenever you want. '[IAbpHostEnvironment](#)' is used by the ABP Framework in several places to perform specific actions by the environment. For example, ABP Framework reduces the cache duration on the \*\*Development\*\* environment for some services.

`'IAbpHostEnvironment'` obtains the current environment name by the following order:

1. Gets and sets the environment name if it's specified in the `'AbpApplicationCreationOptions'`.
  2. Tries to obtain the environment name from the `'IWebHostEnvironment'` or `'IWebAssemblyHostEnvironment'` services for ASP.NET Core & Blazor WASM applications if the environment name isn't specified in the `'AbpApplicationCreationOptions'`.
  3. Sets the environment name as **\*\*Production\*\***, if the environment name is not specified or can not be obtained from the services.

You can configure the `'AbpApplicationCreationOptions'` [options class](Options.md) while creating the ABP application and set an environment name to its `'Environment'` property. You can find the `'AddApplication'` or `'AddApplicationAsync'` call in your solution (typically in the `'Program.cs'` file), and set the `'Environment'` option as shown below:

```
```csharp
await builder.AddApplicationAsync<OrderingServiceHttpApiHostModule>(options
=>
{
    options.Environment = Environments.Staging; //or directly set as
    "Staging"
});
```

Then, whenever you need to get the current environment name or check the environment, you can use the `IAbpHostEnvironment` interface:

```
```csharp
public class MyDemoService
{
 private readonly IAbpHostEnvironment _abpHostEnvironment;

 public MyDemoService(IAbpHostEnvironment abpHostEnvironment)
 {
 _abpHostEnvironment = abpHostEnvironment;
 }

 public void MyMethod()
 {
 var environmentName = _abpHostEnvironment.EnvironmentName;

 if (_abpHostEnvironment.IsDevelopment()) { /* ... */ }

 if (_abpHostEnvironment.IsStaging()) { /* ... */ }

 if (_abpHostEnvironment.IsProduction()) { /* ... */ }

 if (_abpHostEnvironment.IsEnvironment("custom-environment")) { /* ... */
 }
}
```
## .NET Generic Host & ASP.NET Core Integrations
```

`AbpApplicationFactory` can create a standalone ABP application container without any external dependency. However, in most cases, you will want to integrate it with [.NET's generic host](https://learn.microsoft.com/en-us/dotnet/core/extensions/generic-host) or ASP.NET Core. For such usages, ABP provides built-in extension methods to easily create an ABP application container that is well-integrated to these systems.

The [Getting Started with an Empty ASP.NET Core MVC / Razor Pages Application](Getting-Started-AspNetCore-Application.md) document clearly explains how you can create an ABP application container in an ASP.NET Core application.

You can also [create a console application](Startup-Templates/Console) to see how it is integrated with .NET Generic Host.

> Most of the times, you will directly create ABP applications using the ABP CLI's `'new'` command. So, you don't need to care about these integration details.

See Also

- * [Dependency injection](Dependency-Injection.md)
- * [Modularity](Module-Development-Basics.md)

6.2 Authorization

Authorization

Authorization is used to check if a user is allowed to perform some specific operations in the application.

ABP extends [ASP.NET Core Authorization](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction>) by adding **permissions** as auto [policies](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies>) and allowing authorization system to be usable in the **[application services]**(Application-Services.md) too.

So, all the ASP.NET Core authorization features and the documentation are valid in an ABP based application. This document focuses on the features that are added on top of ASP.NET Core authorization features.

Authorize Attribute

ASP.NET Core defines the [**Authorize**](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/simple>) attribute that can be used for an action, a controller or a page. ABP allows you to use the same attribute for an [application service](Application-Services.md) too.

Example:

```
```csharp
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Volo.Abp.Application.Services;

namespace Acme.BookStore
{
 [Authorize]
 public class AuthorAppService : ApplicationService, IAuthorAppService
 {
 public Task<List<AuthorDto>> GetListAsync()
 {
 ...
 }

 [AllowAnonymous]
 public Task<AuthorDto> GetAsync(Guid id)
 {
 ...
 }

 [Authorize("BookStore_Author_Create")]
 public Task CreateAsync(CreateAuthorDto input)
 {
 ...
 }
 }
}
```

```
```
```

- `Authorize` attribute forces the user to login into the application in order to use the `AuthorAppService` methods. So, `GetListAsync` method is only available to the authenticated users.
- `AllowAnonymous` suppresses the authentication. So, `GetAsync` method is available to everyone including unauthorized users.
- `[Authorize("BookStore_Author_Create")]` defines a policy (see [policy based authorization](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies>)) that is checked to authorize the current user.

"BookStore_Author_Create" is an arbitrary policy name. If you declare an attribute like that, ASP.NET Core authorization system expects a policy to be defined before.

You can, of course, implement your policies as stated in the ASP.NET Core documentation. But for simple true/false conditions like a policy was granted to a user or not, ABP defines the permission system which will be explained in the next section.

Permission System

A permission is a simple policy that is granted or prohibited for a particular user, role or client.

Defining Permissions

To define permissions, create a class inheriting from the `PermissionDefinitionProvider` as shown below:

```
```csharp
using Volo.Abp.Authorization.Permissions;

namespace Acme.BookStore.Permissions
{
 public class BookStorePermissionDefinitionProvider :
PermissionDefinitionProvider
 {
 public override void Define(IPermissionDefinitionContext context)
 {
 var myGroup = context.AddGroup("BookStore");

 myGroup.AddPermission("BookStore_Author_Create");
 }
 }
}..
```

> ABP automatically discovers this class. No additional configuration required!

> You typically define this class inside the `Application.Contracts` project of your [application](Startup-Templates/Application.md). The startup template already comes with an empty class named *\*YourProjectNamePermissionDefinitionProvider\** that you can start with.

In the `Define` method, you first need to add a \*\*permission group\*\* or get an existing group then add \*\*permissions\*\* to this group.

When you define a permission, it becomes usable in the ASP.NET Core authorization system as a \*\*policy\*\* name. It also becomes visible in the UI. See permissions dialog for a role:

![authorization-new-permission-ui](images/authorization-new-permission-ui.png)

- The "BookStore" group is shown as a new tab on the left side.
- "BookStore\_Author\_Create" on the right side is the permission name. You can grant or prohibit it for the role.

When you save the dialog, it is saved to the database and used in the authorization system.

> The screen above is available when you have installed the identity module, which is basically used for user and role management. Startup templates come with the identity module pre-installed.

#### #### Localizing the Permission Name

"BookStore\_Author\_Create" is not a good permission name for the UI. Fortunately, `AddPermission` and `AddGroup` methods can take `LocalizableString` as second parameters:

```
```csharp
var myGroup = context.AddGroup(
    "BookStore",
    LocalizableString.Create<BookStoreResource>("BookStore")
);

myGroup.AddPermission(
    "BookStore_Author_Create",

LocalizableString.Create<BookStoreResource>("Permission:BookStore_Author_Crea
te")
);
```

```

Then you can define texts for "BookStore" and "Permission:BookStore\_Author\_Create" keys in the localization file:

```
```json
"BookStore": "Book Store",
"Permission:BookStore_Author_Create": "Creating a new author"
```

```

> For more information, see the [localization document](Localization.md) on the localization system.

The localized UI will be as seen below:

![authorization-new-permission-ui-localized](images/authorization-new-permission-ui-localized.png)

#### #### Multi-Tenancy

ABP supports [multi-tenancy](Multi-Tenancy.md) as a first class citizen. You can define multi-tenancy side option while defining a new permission. It gets one of the three values defined below:

- \*\*Host\*\*: The permission is available only for the host side.
- \*\*Tenant\*\*: The permission is available only for the tenant side.
- \*\*Both\*\* (default): The permission is available both for tenant and host sides.

> If your application is not multi-tenant, you can ignore this option.

To set the multi-tenancy side option, pass to the third parameter of the `AddPermission` method:

```
```csharp
myGroup.AddPermission(
    "BookStore_Author_Create",
    LocalizableString.Create<BookStoreResource>("Permission:BookStore_Author_Crea
te"),
    multiTenancySide: MultiTenancySides.Tenant //set multi-tenancy side!
);
```

Enable/Disable Permissions

A permission is enabled by default. It is possible to disable a permission. A disabled permission will be prohibited for everyone. You can still check for the permission, but it will always return prohibited.

Example definition:

```
```csharp
myGroup.AddPermission("Author_Management", isEnabled: false);
```
```

You normally don't need to define a disabled permission (unless you temporary want disable a feature of your application). However, you may want to disable a permission defined in a depended module. In this way you can disable the related application functionality. See the "**Changing Permission Definitions of a Depended Module**" section below for an example usage.

> Note: Checking an undefined permission will throw an exception while a disabled permission check simply returns prohibited (false).

Child Permissions

A permission may have child permissions. It is especially useful when you want to create a hierarchical permission tree where a permission may have additional sub permissions which are available only if the parent permission has been granted.

Example definition:

```
```csharp
var authorManagement = myGroup.AddPermission("Author_Management");
authorManagement.AddChild("Author_Management_Create_Books");
```

```
authorManagement.AddChild("Author_Management_Edit_Books");
authorManagement.AddChild("Author_Management_Delete_Books");
````
```

The result on the UI is shown below (you probably want to localize permissions for your application):

![authorization-new-permission-ui-hierarchy](images/authorization-new-permission-ui-hierarchy.png)

For the example code, it is assumed that a role/user with "Author_Management" permission granted may have additional permissions. Then a typical application service that checks permissions can be defined as shown below:

```
```csharp
[Authorize("Author_Management")]
public class AuthorAppService : ApplicationService, IAuthorAppService
{
 public Task<List<AuthorDto>> GetListAsync()
 {
 ...
 }

 public Task<AuthorDto> GetAsync(Guid id)
 {
 ...
 }

 [Authorize("Author_Management_Create_Books")]
 public Task CreateAsync(CreateAuthorDto input)
 {
 ...
 }

 [Authorize("Author_Management_Edit_Books")]
 public Task UpdateAsync(CreateAuthorDto input)
 {
 ...
 }

 [Authorize("Author_Management_Delete_Books")]
 public Task DeleteAsync(CreateAuthorDto input)
 {
 ...
 }
}
````
```

- `GetListAsync` and `GetAsync` will be available to users if they have `Author_Management` permission is granted.
- Other methods require additional permissions.

Overriding a Permission by a Custom Policy

If you define and register a policy to the ASP.NET Core authorization system with the same name of a permission, your policy will override the existing permission. This is a powerful way to extend the authorization for a pre-built module that you are using in your application.

See [policy based authorization](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/policies>) document to learn how to define a custom policy.

Changing Permission Definitions of a Depended Module

A class deriving from the `PermissionDefinitionProvider` (just like the example above) can also get existing permission definitions (defined by the depended [modules](Module-Development-Basics.md)) and change their definitions.

Example:

```
```csharp
context
 .GetPermissionOrNull(IdentityPermissions.Roles.Delete)
 .IsEnabled = false;
```
```

When you write this code inside your permission definition provider, it finds the "role deletion" permission of the [Identity Module](Modules/Identity.md) and disabled the permission, so no one can delete a role on the application.

> Tip: It is better to check the value returned by the `GetPermissionOrNull` method since it may return null if the given permission was not defined.

Permission Depending on a Condition

You may want to disable a permission based on a condition. Disabled permissions are not visible on the UI and always returns `prohibited` when you check them. There are two built-in conditional dependencies for a permission definition;

- * A permission can be automatically disabled if a [Feature](Features.md) was disabled.
- * A permission can be automatically disabled if a [Global Feature](Global-Features.md) was disabled.

In addition, you can create your custom extensions.

Depending on a Features

Use the `RequireFeatures` extension method on your permission definition to make the permission available only if a given feature is enabled:

```
```csharp
myGroup.AddPermission("Book_Creation")
 .RequireFeatures("BookManagement");
```
```

Depending on a Global Feature

Use the `RequireGlobalFeatures` extension method on your permission definition to make the permission available only if a given feature is enabled:

```
```csharp
```

```
myGroup.AddPermission("Book_Creation")
 .RequireGlobalFeatures("BookManagement");
```
#### Creating a Custom Permission Dependency

`'PermissionDefinition` supports state check, Please refer to [Simple State Checker's documentation](SimpleStateChecker.md)
```

IAuthorizationService

ASP.NET Core provides the `IAuthorizationService` that can be used to check for authorization. Once you inject, you can use it in your code to conditionally control the authorization.

Example:

```
```csharp
public async Task CreateAsync(CreateAuthorDto input)
{
 var result = await AuthorizationService
 .AuthorizeAsync("Author_Management_Create_Books");
 if (result.Succeeded == false)
 {
 //throw exception
 throw new AbpAuthorizationException("...");
 }

 //continue to the normal flow...
}
```

```

> `AuthorizationService` is available as a property when you derive from ABP's `ApplicationService` base class. Since it is widely used in application services, `ApplicationService` pre-injects it for you. Otherwise, you can directly [inject](Dependency-Injection.md) it into your class.

Since this is a typical code block, ABP provides extension methods to simplify it.

Example:

```
```csharp
public async Task CreateAsync(CreateAuthorDto input)
{
 await AuthorizationService.CheckAsync("Author_Management_Create_Books");

 //continue to the normal flow...
}
```

```

`CheckAsync` extension method throws `AbpAuthorizationException` if the current user/client is not granted for the given permission. There is also `IsGrantedAsync` extension method that returns `true` or `false`.

`IAuthorizationService` has some overloads for the `AuthorizeAsync` method. These are explained in the [ASP.NET Core authorization

[documentation](https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction)](<https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction>).

> Tip: Prefer to use the `Authorize` attribute wherever possible, since it is declarative & simple. Use `IAuthorizationService` if you need to conditionally check a permission and run a business code based on the permission check.

Check a Permission in JavaScript

See the following documents to learn how to re-use the authorization system on the client side:

- * [ASP.NET Core MVC / Razor Pages UI: [Authorization](#)](UI/AspNetCore/JavaScript-API/Auth.md)
- * [Angular UI Authorization](UI/Angular/Permission-Management.md)
- * [Blazor UI Authorization](UI/Blazor/Authorization.md)

Permission Management

Permission management is normally done by an admin user using the permission management modal:

![authorization-new-permission-ui-localized](images/authorization-new-permission-ui-localized.png)

If you need to manage permissions by code, inject the `IPermissionManager` and use as shown below:

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IPermissionManager _permissionManager;

 public MyService(IPermanentManager permanentManager)
 {
 _permanentManager = permanentManager;
 }

 public async Task GrantPermissionForUserAsync(Guid userId, string
 permissionName)
 {
 await _permanentManager.SetForUserAsync(userId, permissionName,
 true);
 }

 public async Task ProhibitPermissionForUserAsync(Guid userId, string
 permissionName)
 {
 await _permanentManager.SetForUserAsync(userId, permissionName,
 false);
 }
}..
```

`SetForUserAsync` sets the value (true/false) for a permission of a user. There are more extension methods like `SetForRoleAsync` and `SetForClientAsync`.

`IPermissionManager` is defined by the permission management module. See the [permission management module documentation](Modules/PermissionManagement.md) for more information.

## ## Advanced Topics

### ### Permission Value Providers

Permission checking system is extensible. Any class derived from `PermissionValueProvider` (or implements `IPermissionValueProvider`) can contribute to the permission check. There are three pre-defined value providers:

- `UserPermissionValueProvider` checks if the current user is granted for the given permission. It gets user id from the current claims. User claim name is defined with the `AbpClaimTypes.UserId` static property.
- `RolePermissionValueProvider` checks if any of the roles of the current user is granted for the given permission. It gets role names from the current claims. Role claims name is defined with the `AbpClaimTypes.Role` static property.
- `ClientPermissionValueProvider` checks if the current client is granted for the given permission. This is especially useful on a machine to machine interaction where there is no current user. It gets the client id from the current claims. Client claim name is defined with the `AbpClaimTypes.ClientId` static property.

You can extend the permission checking system by defining your own permission value provider.

Example:

```
```csharp
public class SystemAdminPermissionValueProvider : PermissionValueProvider
{
    public SystemAdminPermissionValueProvider(IPermissionStore
permissionStore)
        : base(permissionStore)
    {

    }

    public override string Name => "SystemAdmin";

    public async override Task<PermissionGrantResult>
        CheckAsync(PermissionValueCheckContext context)
    {
        if (context.Principal?.FindFirst("User_Type")?.Value ==
"SystemAdmin")
        {
            return PermissionGrantResult.Granted;
        }

        return PermissionGrantResult.Undefined;
    }
}
```

```

This provider allows for all permissions to a user with a `User\_Type` claim that has `SystemAdmin` value. It is common to use current claims and `IPermissionStore` in a permission value provider.

A permission value provider should return one of the following values from the `CheckAsync` method:

- `PermissionGrantResult.Granted` is returned to grant the user for the permission. If any of the providers return `Granted`, the result will be `Granted`, if no other provider returns `Prohibited`.
- `PermissionGrantResult.Prohibited` is returned to prohibit the user for the permission. If any of the providers return `Prohibited`, the result will always be `Prohibited`. Doesn't matter what other providers return.
- `PermissionGrantResult.Undefined` is returned if this value provider could not decide about the permission value. Return this to let other providers check the permission.

Once a provider is defined, it should be added to the `AbpPermissionOptions` as shown below:

```
```csharp
Configure<AbpPermissionOptions>(options =>
{
    options.ValueProviders.Add<SystemAdminPermissionValueProvider>();
});
```

Permission Store

`IPermissionStore` is the only interface that needs to be implemented to read the value of permissions from a persistence source, generally a database system. The Permission Management module implements it and pre-installed in the application startup template. See the [permission management module documentation](Modules/Permission-Management.md) for more information

AlwaysAllowAuthorizationService

`AlwaysAllowAuthorizationService` is a class that is used to bypass the authorization service. It is generally used in integration tests where you may want to disable the authorization system.

Use `IServiceCollection.AddAlwaysAllowAuthorization()` extension method to register the `AlwaysAllowAuthorizationService` to the [dependency injection](Dependency-Injection.md) system:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 context.Services.AddAlwaysAllowAuthorization();
}
```

This is already done for the startup template integration tests.

### ### Claims Principal Factory

Claims are important elements of authentication and authorization. ABP uses the `IAbpClaimsPrincipalFactory` service to create claims on authentication.

This service was designed as extensible. If you need to add your custom claims to the authentication ticket, you can implement the `IAbpClaimsPrincipalContributor` in your application.

\*\*Example: Add a `SocialSecurityNumber` claim and get it:\*\*

```
```csharp
public class SocialSecurityNumberClaimsPrincipalContributor : 
IAbpClaimsPrincipalContributor, ITransientDependency
{
    public async Task ContributeAsync(AbpClaimsPrincipalContributorContext
context)
    {
        var identity = context.CclaimsPrincipal.Identities.FirstOrDefault();
        var userId = identity?.FindUserId();
        if (userId.HasValue)
        {
            var userService =
context.ServiceProvider.GetRequiredService<IUserService>(); //Your custom
service
            var socialSecurityNumber = await
userService.GetSocialSecurityNumberAsync(userId.Value);
            if (socialSecurityNumber != null)
            {
                identity.AddClaim(new Claim("SocialSecurityNumber",
socialSecurityNumber));
            }
        }
    }
}

public static class CurrentUserExtensions
{
    public static string GetSocialSecurityNumber(this ICurrentUser
currentUser)
    {
        return currentUser.FindClaimValue("SocialSecurityNumber");
    }
}..
```

> If you use Identity Server please add your claims to `RequestedClaims` of `AbpClaimsServiceOptions` .

```
```csharp
Configure<AbpClaimsServiceOptions>(options =>
{
 options.RequestedClaims.AddRange(new[] { "SocialSecurityNumber" });
});
```

## See Also

- \* [Permission Management Module](Modules/Permission-Management.md)
- \* [ASP.NET Core MVC / Razor Pages JavaScript Auth API](UI/AspNetCore/JavaScript-API/Auth.md)
- \* [Permission Management in Angular UI](UI/Angular/Permission-Management.md)

## 6.3 Caching

### # Distributed Caching

ABP Framework extends the [ASP.NET Core distributed cache](<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>).

> \*\*Default implementation of the `IDistributedCache` interface is `MemoryDistributedCache` which works in-memory.\*\* See [ASP.NET Core's documentation](<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>) to see how to switch to Redis or another cache provider. Also, see the [Redis Cache](Redis-Cache.md) document if you want to use Redis as the distributed cache server.

### ## Installation

> This package is already installed by default with the [application startup template](Startup-Templates/Application.md). So, most of the time, you don't need to install it manually.

[Volo.Abp.Caching](<https://www.nuget.org/packages/Volo.Abp.Caching>) is the main package of the caching system. You can install it a project using the add-package command of the [ABP CLI](CLI.md):

```
```bash
abp add-package Volo.Abp.Caching
````
```

You need to run this command on a command line terminal in a folder containing a `csproj` file (see [other options](<https://abp.io/package-detail/Volo.Abp.Caching>) to install).

### ## Usage

#### ### `IDistributedCache` Interface

ASP.NET Core defines the `IDistributedCache` interface to get/set the cache values. But it has some difficulties:

- \* It works with \*\*byte arrays\*\* rather than .NET objects. So, you need to \*\*serialize/deserialize\*\* the objects you need to cache.
- \* It provides a \*\*single key pool\*\* for all cache items, so;
  - \* You need to care about the keys to distinguish \*\*different type of objects\*\*.
  - \* You need to care about the cache items of \*\*different tenants\*\* in a [multi-tenant](Multi-Tenancy.md) system.

> `IDistributedCache` is defined in the `Microsoft.Extensions.Caching.Abstractions` package. That means it is not only usable for ASP.NET Core applications, but also available to \*\*any type of applications\*\*.

See [ASP.NET Core's distributed caching document](<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>) for more information.

### ### `IDistributedCache<TCacheItem>` Interface

ABP framework defines the generic `IDistributedCache<TCacheItem>` interface in the [Volo.Abp.Caching](<https://www.nuget.org/packages/Volo.Abp.Caching/>) package. `TCacheItem` is the type of the object stored in the cache.

`IDistributedCache<TCacheItem>` solves the difficulties explained above;

- \* It internally \*\*serializes/deserializes\*\* the cached objects. Uses \*\*JSON\*\* serialization by default, but can be overridden by replacing the `IDistributedCacheSerializer` service in the [dependency injection](Dependency-Injection.md) system.
- \* It automatically adds a \*\*cache name\*\* prefix to the cache keys based on the object type stored in the cache. Default cache name is the full name of the cache item class (`CacheItem` postfix is removed if your cache item class ends with it). You can use the \*\*`CacheName` attribute\*\* on the cache item class to set the cache name.
- \* It automatically adds the \*\*current tenant id\*\* to the cache key to distinguish cache items for different tenants (if your application is [multi-tenant](Multi-Tenancy.md)). Define `IgnoreMultiTenancy` attribute on the cache item class to disable this if you want to share the cached objects among all tenants in a multi-tenant application.
- \* Allows to define a \*\*global cache key prefix\*\* per application, so different applications can use their isolated key pools in a shared distributed cache server.
- \* It \*\*can tolerate errors\*\* wherever possible and bypasses the cache. This is useful when you have temporary problems on the cache server.
- \* It has methods like `GetManyAsync` and `SetManyAsync` which significantly improve the performance on \*\*batch operations\*\*.

\*\*Example: Store Book names and prices in the cache\*\*

```
```csharp
namespace MyProject
{
    public class BookCacheItem
    {
        public string Name { get; set; }

        public float Price { get; set; }
    }
}
```

You can inject and use the `IDistributedCache<BookCacheItem>` service to get/set `BookCacheItem` objects:

```
```csharp
using System;
using System.Threading.Tasks;
using Microsoft.Extensions.Caching.Distributed;
using Volo.Abp.Caching;
using Volo.Abp.DependencyInjection;
```

```

namespace MyProject
{
 public class BookService : ITransientDependency
 {
 private readonly IDistributedCache<BookCacheItem> _cache;

 public BookService(IDistributedCache<BookCacheItem> cache)
 {
 _cache = cache;
 }

 public async Task<BookCacheItem> GetAsync(Guid bookId)
 {
 return await _cache.GetOrAddAsync(
 bookId.ToString(), //Cache key
 async () => await GetBookFromDatabaseAsync(bookId),
 () => new DistributedCacheEntryOptions
 {
 AbsoluteExpiration = DateTimeOffset.Now.AddHours(1)
 }
);
 }

 private Task<BookCacheItem> GetBookFromDatabaseAsync(Guid bookId)
 {
 //TODO: get from database
 }
 }
}

* This sample service uses the `GetOrAddAsync()` method to get a book item
from the cache. `GetOrAddAsync` is an additional method that was added by the
ABP Framework to the standard ASP.NET Core distributed cache methods.
* If the book was not found in the cache, it calls the factory method
(`GetBookFromDatabaseAsync` in this case) to retrieve the book item from the
original source.
* `GetOrAddAsync` optionally gets a `DistributedCacheEntryOptions` which can
be used to set the lifetime of the cached item.

`IDistributedCache<BookCacheItem>` supports the same methods of the ASP.NET
Core's standard `IDistributedCache` interface, so you can refer [it's
documentation](https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed).

`IDistributedCache<TCacheItem, TCacheKey>` Interface

`IDistributedCache<TCacheItem>` interface assumes that the type of your
cache key is `string` (so, you need to manually convert your key to
string if you need to use a different kind of cache key). While this is not a
big deal, `IDistributedCache<TCacheItem, TCacheKey>` can be used when your
cache key type is not `string`.

Example: Store Book names and prices in the cache

```csharp
using Volo.Abp.Caching;
```

```

```

namespace MyProject
{
 [CacheName("Books")]
 public class BookCacheItem
 {
 public string Name { get; set; }

 public float Price { get; set; }
 }
}
...

```

\* This example uses the `CacheName` attribute for the `BookCacheItem` class to set the cache name.

You can inject and use the `IDistributedCache<BookCacheItem, Guid>` service to get/set `BookCacheItem` objects:

```

```csharp
using System;
using System.Threading.Tasks;
using Microsoft.Extensions.Caching.Distributed;
using Volo.Abp.Caching;
using Volo.Abp.DependencyInjection;

namespace MyProject
{
    public class BookService : ITransientDependency
    {
        private readonly IDistributedCache<BookCacheItem, Guid> _cache;

        public BookService(IDistributedCache<BookCacheItem, Guid> cache)
        {
            _cache = cache;
        }

        public async Task<BookCacheItem> GetAsync(Guid bookId)
        {
            return await _cache.GetOrAddAsync(
                bookId, //Guid type used as the cache key
                async () => await GetBookFromDatabaseAsync(bookId),
                () => new DistributedCacheEntryOptions
                {
                    AbsoluteExpiration = DateTimeOffset.Now.AddHours(1)
                }
            );
        }

        private Task<BookCacheItem> GetBookFromDatabaseAsync(Guid bookId)
        {
            //TODO: get from database
        }
    }
}
...

```

* This sample service uses the `GetOrAddAsync()` method to get a book item from the cache.

* Since cache explicitly implemented as using `Guid` as cache key, `Guid` value passed to `_cache_GetOrAddAsync()` method.

Complex Types as the Cache Key

`IDistributedCache<TCacheItem, TCacheKey>` internally uses `ToString()` method of the key object to convert it to a string. If you need to use a complex object as the cache key, you need to override `ToString` method of your class.

An example class that is used as a cache key:

```
```csharp
public class UserInOrganizationCacheKey
{
 public Guid UserId { get; set; }

 public Guid OrganizationId { get; set; }

 //Builds the cache key
 public override string ToString()
 {
 return $"{UserId}_{OrganizationId}";
 }
}
````
```

Example usage:

```
```csharp
public class BookService : ITransientDependency
{
 private readonly IDistributedCache<UserCacheItem,
UserInOrganizationCacheKey> _cache;

 public BookService(
 IDistributedCache<UserCacheItem, UserInOrganizationCacheKey> cache)
 {
 _cache = cache;
 }

 ...
}

Configuration

AbpDistributedCacheOptions

`AbpDistributedCacheOptions` is the main [options class](Options.md) to
configure the caching.

Example: Set the cache key prefix for the application

```csharp
Configure<AbpDistributedCacheOptions>(options =>
{
    options.KeyPrefix = "MyApp1";
}
````
```

```
});

> Write that code inside the `ConfigureServices` method of your [module class](Module-Development-Basics.md).

Available Options

* `HideErrors` (`bool`, default: `true`): Enables/disables hiding the errors on writing/reading values from the cache server.
* `KeyPrefix` (`string`, default: `null`): If your cache server is shared by multiple applications, you can set a prefix for the cache keys for your application. In this case, different applications can not overwrite each other's cache items.
* `GlobalCacheEntryOptions` (`DistributedCacheEntryOptions`): Used to set default distributed cache options (like `AbsoluteExpiration` and `SlidingExpiration`) used when you don't specify the options while saving cache items. Default value uses the `SlidingExpiration` as 20 minutes.
```

## ## Error Handling

When you design a cache for your objects, you typically try to get the value from cache first. If not found in the cache, you query the object from the \*\*original source\*\*. It may be located in a \*\*database\*\* or may require to perform an HTTP call to a remote server.

In most cases, you want to \*\*tolerate the cache errors\*\*; If you get error from the cache server you don't want to cancel the operation. Instead, you silently hide (and log) the error and \*\*query from the original source\*\*. This is what the ABP Framework does by default.

ABP's Distributed Cache [handle](Exception-Handling.md), log and hide errors by default. There is an option to change this globally (see the options below).

In addition, all of the `IDistributedCache<TCacheItem>` (and `IDistributedCache<TCacheItem, TCacheKey>`) methods have an optional `hideErrors` parameter, which is `null` by default. The global value is used if this parameter left as `null`, otherwise you can decide to hide or throw the exceptions for individual method calls.

## ## Batch Operations

ABP's distributed cache interfaces provide methods to perform batch methods those improves the performance when you want to batch operation multiple cache items in a single method call.

- \* `SetManyAsync` and `SetMany` methods can be used to set multiple values to the cache.
- \* `GetManyAsync` and `GetMany` methods can be used to retrieve multiple values from the cache.
- \* `GetOrAddManyAsync` and `GetOrAddMany` methods can be used to retrieve multiple values and set missing values from the cache
- \* `RefreshManyAsync` and `RefreshMany` methods can be used to resets the sliding expiration timeout of multiple values from the cache
- \* `RemoveManyAsync` and `RemoveMany` methods can be used to remove multiple values from the cache

> These are not standard methods of the ASP.NET Core caching. So, some providers may not support them. They are supported by the [ABP Redis Cache integration package](Redis-Cache.md). If the provider doesn't support, it fallbacks to 'SetAsync' and 'GetAsync' ... methods (called once for each item).

## ## Caching Entities

ABP Framework provides a [Distributed Entity Cache System](Entity-Cache.md) for caching entities. It is useful if you want to use caching for quicker access to the entity rather than repeatedly querying it from the database.

It's designed as read-only and automatically invalidates a cached entity if the entity is updated or deleted.

> See the [Entity Cache](Entity-Cache.md) documentation for more information.

## ## Advanced Topics

### ### Unit Of Work Level Cache

Distributed cache service provides an interesting feature. Assume that you've updated the price of a book in the database, then set the new price to the cache, so you can use the cached value later. What if you have an exception after setting the cache and you **rollback the transaction** that updates the price of the book? In this case, cache value will be incorrect.

'`IDistributedCache<..>`' methods gets an optional parameter, named '`considerUow`', which is '`false`' by default. If you set it to '`true`', then the changes you made for the cache are not actually applied to the real cache store, but associated with the current [unit of work](Unit-Of-Work.md). You get the value you set in the same unit of work, but the changes are applied **only if the current unit of work succeed**.

### ### IDistributedCacheSerializer

'`IDistributedCacheSerializer`' service is used to serialize and deserialize the cache items. Default implementation is the '`Utf8JsonDistributedCacheSerializer`' class that uses '`IJsonSerializer`' service to convert objects to [JSON](Json-Serialization.md) and vice versa. Then it uses UTC8 encoding to convert the JSON string to a byte array which is accepted by the distributed cache.

You can [replace](Dependency-Injection.md) this service by your own implementation if you want to implement your own serialization logic.

### ### IDistributedCacheKeyNormalizer

'`IDistributedCacheKeyNormalizer`' is implemented by the '`DistributedCacheKeyNormalizer`' class by default. It adds cache name, application cache prefix and current tenant id to the cache key. If you need a more advanced key normalization, you can [replace](Dependency-Injection.md) this service by your own implementation.

## ## See Also

- \* [Entity Cache](Entity-Cache.md)
- \* [Redis Cache](Redis-Cache.md)

### 6.3.1 Entity Cache

#### # Entity Cache

ABP Framework provides an entity caching system that works on top of the [distributed caching](Caching.md) system. It does the following operations on behalf of you:

- \* Gets the entity from the database (by using the [repositories](Repositories.md)) in its first call and then gets it from the cache in subsequent calls.
- \* Automatically invalidates the cached entity if the entity is updated or deleted. Thus, it will be retrieved from the database in the next call and will be re-cached.

#### ## Caching Entity Objects

`IEntityCache< TEntityCacheItem, TKey>` is a simple service provided by the ABP Framework for caching entities. Assume that you have a `Product` entity as shown below:

```
```csharp
public class Product : AggregateRoot<Guid>
{
    public string Name { get; set; }
    public string Description { get; set; }
    public float Price { get; set; }
    public int StockCount { get; set; }
}
```

```

If you want to cache this entity, you should first configure the [dependency injection](Dependency-Injection.md) system to register the `IEntityCache` service in the `ConfigureServices` method of your [module class](Module-Development-Basics.md):

```
```csharp
context.Services.AddEntityCache<Product, Guid>();
```

```

Now you can inject the `IEntityCache<Product, Guid>` service wherever you need:

```
```csharp
public class ProductAppService : ApplicationService, IProductAppService
{
    private readonly IEntityCache<Product, Guid> _productCache;

    public ProductAppService(IEntityCache<Product, Guid> productCache)
    {
        _productCache = productCache;
    }

    public async Task<ProductDto> GetAsync(Guid id)
    {
        var product = await _productCache.GetAsync(id);
    }
}
```

```

```
 return ObjectMapper.Map<Product, ProductDto>(product);
 }
}..
```

> Note that we've used the `ObjectMapper` service to map from `Product` to `ProductDto`. You should configure that [object mapping](Object-To-Object-Mapping.md) to make that example service properly works.

That's all. The cache name (in the distributed cache server) will be full name (with namespace) of the `Product` class. You can use the `[CacheName]` attribute to change it. Please refer to the [caching document](Caching.md) for details.

## ## Using a Cache Item Class

In the previous section, we've directly cached the `Product` entity. In that case, the `Product` class must be serializable to JSON (and deserializable from JSON). Sometimes that might not be possible or you may want to use another class to store the cache data. For example, we may want to use the `ProductDto` class instead of the `Product` class for the cached object if the `Product` entity.

Assume that we've created a `ProductDto` class as shown below:

```
```csharp
public class ProductDto : EntityDto<Guid>
{
    public string Name { get; set; }
    public string Description { get; set; }
    public float Price { get; set; }
    public int StockCount { get; set; }
}..
```

Now, we can register the entity cache services to [dependency injection](Dependency-Injection.md) in the `ConfigureServices` method of your [module class](Module-Development-Basics.md) with three generic parameters, as shown below:

```
```csharp
context.Services.AddEntityCache<Product, ProductDto, Guid>();
```
```

Since the entity cache system will perform the [object mapping](Object-To-Object-Mapping.md) (from `Product` to `ProductDto`), we should configure the object map. Here, an example configuration with [AutoMapper](<https://automapper.org/>):

```
```csharp
public class MyMapperProfile : Profile
{
 public MyMapperProfile()
 {
 CreateMap<Product, ProductDto>();
 }
}..
```

Now, you can inject the `IEntityCache<ProductDto, Guid>` service wherever you want:

```
```csharp
public class ProductAppService : ApplicationService, IProductAppService
{
    private readonly IEntityCache<ProductDto, Guid> _productCache;

    public ProductAppService(IEntityCache<ProductDto, Guid> productCache)
    {
        _productCache = productCache;
    }

    public async Task<ProductDto> GetAsync(Guid id)
    {
        return await _productCache.GetAsync(id);
    }
}..
```

Notice that the `._productCache.GetAsync` method already returns a `ProductDto` object, so we could directly return it from our application service.

Configuration

All of the `context.Services.AddEntityCache()` methods get an optional `DistributedCacheEntryOptions` parameter where you can easily configure the caching options:

```
```csharp
context.Services.AddEntityCache<Product, ProductDto, Guid>(
 new DistributedCacheEntryOptions
 {
 SlidingExpiration = TimeSpan.FromMinutes(30)
 }
);
```

> The default cache duration is \*\*2 minutes\*\* with the `AbsoluteExpirationRelativeToNow` configuration.

## ## Additional Notes

- \* Entity classes should be serializable/deserializable to/from JSON to be cached (because it's serialized to JSON when saving in the [Distributed Cache](Caching.md)). If your entity class is not serializable, you can consider using a cache-item/DTO class instead, as explained before.
- \* Entity Caching System is designed as \*\*read-only\*\*. You should use the standard [repository](Repositories.md) methods to manipulate the entity if you need. If you need to manipulate (update) the entity, do not get it from the entity cache. Instead, read it from the repository, change it and update using the repository.

## ## See Also

- \* [Distributed caching](Caching.md)

```
* [Entities](Entities.md)
* [Repositories](Repositories.md)
```

### 6.3.2 Redis Cache

#### # Redis Cache

ABP Framework [Caching System](Caching.md) extends the [ASP.NET Core distributed cache](<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>). So, \*\*any provider\*\* supported by the standard ASP.NET Core distributed cache can be usable in your application and can be configured just like \*\*documented by Microsoft\*\*.

However, ABP provides an \*\*integration package\*\* for Redis Cache: [Volo.Abp.Caching.StackExchangeRedis](<https://www.nuget.org/packages/Volo.Abp.Caching.StackExchangeRedis>). There are two reasons for using this package, instead of the standard [Microsoft.Extensions.Caching.StackExchangeRedis](<https://www.nuget.org/packages/Microsoft.Extensions.Caching.StackExchangeRedis/>) package.

1. It implements `SetManyAsync` and `GetManyAsync` methods. These are not standard methods of the Microsoft Caching library, but added by the ABP Framework [Caching](Caching.md) system. They \*\*significantly increases the performance\*\* when you need to set/get multiple cache items with a single method call.
2. It \*\*simplifies\*\* the Redis cache \*\*configuration\*\* (will be explained below).

> Volo.Abp.Caching.StackExchangeRedis is already uses the Microsoft.Extensions.Caching.StackExchangeRedis package, but extends and improves it.

#### ## Installation

> This package is already installed in the application startup template if it is using Redis.

Open a command line in the folder of your `'.csproj'` file and type the following ABP CLI command:

```
```bash
abp add-package Volo.Abp.Caching.StackExchangeRedis
````
```

#### ## Configuration

Volo.Abp.Caching.StackExchangeRedis package automatically gets the Redis [configuration](Configuration.md) from the `IConfiguration`. So, for example, you can set your configuration inside the `appsettings.json`:

```
```js
"Redis": {
  "Enabled": "true",
  "Configuration": "127.0.0.1"
}
````
```

The setting '`.IsEnabled`' is optional and will be considered '`true`' if it is not set.

Alternatively you can configure the standard `[RedisCacheOptions]`(<https://docs.microsoft.com/en-us/dotnet/api/microsoft.extensions.caching.stackexchangedis.rediscacheoptions>) `[options]`(`Options.md`) class in the '`ConfigureServices`' method of your `[module]`(`Module-Development-Basics.md`):

```
```csharp
Configure<RedisCacheOptions>(options =>
{
    //...
});
```

See Also

- * `[Caching]`(`Caching.md`)

6.4 Configuration

Configuration

ASP.NET Core has an flexible and extensible key-value based configuration system. In fact, the configuration system is a part of `Microsoft.Extensions` libraries and it is independent from ASP.NET Core. That means it can be used in any type of application. See `[Microsoft's documentation]`(<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/>) to learn the configuration infrastructure. ABP framework is 100% compatible with the configuration system.

6.5 Connection Strings

Connection Strings

> Connection string system is especially needed when you want to create or use a modular system. If you have a monolithic application with a single database, you can go with the `[ABP startup solution template]`(`Startup-Templates/Application.md`), which is properly configured for you.

ABP Framework is designed to be `[modular]`(`Module-Development-Basics.md`) and `[multi-tenancy]`(`Multi-Tenancy.md`) aware. Connection string management is also designed to support these scenarios;

- * Allows to set separate connection strings for every module, so every module can have its own physical database. Modules even might be configured to use different database providers.
- * Allows to set separate connection string and use a separate database per tenant (in a SaaS application).

It also supports hybrid scenarios;

- * Allows to group modules into databases (e.g., all modules into a single shared database or two modules to database A, three modules to database B, one module to database C and rest of the modules to database D)
- * Allows to group tenants into databases, just like the modules.
- * Allows to separate databases per tenant per module (which might be hard to maintain for you because of too many databases, but the ABP framework supports it).

All the [pre-built application modules](Modules/Index.md) are designed to be compatible these scenarios.

`## Configure the Connection Strings`

See the following configuration:

```
```json
"ConnectionStrings": {
 "Default": "Server=localhost;Database=MyMainDb;Trusted_Connection=True;",
 "AbpIdentityServer": "Server=localhost;Database=MyIdsDb;Trusted_Connection=True;",
 "AbpPermissionManagement": "Server=localhost;Database=MyPermissionDb;Trusted_Connection=True;"
}
```

```

> ABP uses the `IConfiguration` service to get the application configuration. While the simplest way to write configuration into the `appsettings.json` file, it is not limited to this file. You can use environment variables, user secrets, Azure Key Vault... etc. See the [configuration](Configuration.md) document for more.

This configuration defines three different connection strings:

- * `MyMainDb` (the `Default` connection string) is the main connection string of the application. If you don't specify a connection string for a module, it fallbacks to the `Default` connection string. The [application startup template](Startup-Templates/Application.md) is configured to use a single connection string, so all the modules uses a single, shared database.
- * `MyIdsDb` (the `AbpIdentityServer` connection string) is used by the [IdentityServer](Modules/IdentityServer.md) module.
- * `MyPermissionDb` (the `AbpPermissionManagement` connection string) is used by the [Permission Management](Modules/Permission-Management.md) module.

[Pre-built application modules](Modules/Index.md) define constants for the connection string names. For example, the [IdentityServer module](Modules/IdentityServer.md) defines a `ConnectionStringName` constant in the `AbpIdentityServerDbProperties` class (located in the `Volo.Abp.IdentityServer` namespace). Other modules similarly define constants, so you can investigate the connection string name.

`### AbpDbConnectionOptions`

`AbpDbConnectionOptions` is the options class that is used to set the connection strings and configure database structures.

`#### Setting the connection strings`

ABP uses the `AbpDbConnectionOptions` to get the connection strings. If you configure the connection strings as explained above, `AbpDbConnectionOptions` is automatically filled. However, you can set or override the connection strings using [the options pattern](Options.md). You can configure the `AbpDbConnectionOptions` in the `ConfigureServices` method of your [module](Module-Development-Basics.md) as shown below:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 Configure<AbpDbConnectionOptions>(options =>
 {
 options.ConnectionStrings.Default = "...";
 options.ConnectionStrings["AbpPermissionManagement"] = "...";
 });
}
````
```

Configuring the database structures

`Databases` property of the `AbpDbConnectionOptions` class is used to group multiple connection strings (of multiple modules) to a single connection string.

See the following connection strings:

```
```json
"ConnectionStrings": {
 "Default": "Server=localhost;Database=MyMainDb;Trusted_Connection=True;",
 "AbpIdentity": "Server=localhost;Database=MySecondaryDb;Trusted_Connection=True;",
 "AbpIdentityServer": "Server=localhost;Database=MySecondaryDb;Trusted_Connection=True;",
 "AbpPermissionManagement": "Server=localhost;Database=MySecondaryDb;Trusted_Connection=True;"
}
````
```

In this example, we've defined four connection strings, but the last three of them are the same; `AbpIdentity`, `AbpIdentityServer` and `AbpPermissionManagement` uses the same database, named `MySecondaryDb`. The main application and the other modules use the `Default` connection string, hence the `MyMainDb` database.

What we want to do here is to group three modules (`AbpIdentity`, `AbpIdentityServer` and `AbpPermissionManagement`) in a single database, but we needed to specify each one manually. Because the fallback connection string is the `Default` one, if we don't specify it for a module.

To eliminate the repetitive connection string definition, we can configure the `AbpDbConnectionOptions.Databases` property to group these connection strings, as shown in the following code (we place that in the `ConfigureServices` method of our [module class](Module-Development-Basics.md)):

```
```csharp
Configure<AbpDbConnectionOptions>(options =>
{
````
```

```

        options.Databases.Configure("MySecondaryDb", db =>
    {
        db.MappedConnections.Add("AbpIdentity");
        db.MappedConnections.Add("AbpIdentityServer");
        db.MappedConnections.Add("AbpPermissionManagement");
    });
});
```

```

Then we can change the `'appsettings.json'` file as shown in the following code block:

```

```json
"ConnectionStrings": {
    "Default": "Server=localhost;Database=MyMainDb;Trusted_Connection=True;",
    "MySecondaryDb":
    "Server=localhost;Database=MySecondaryDb;Trusted_Connection=True;"
}
```

```

`'MySecondaryDb'` becomes the new connection string for the mapped connections.

> ABP first looks for the module-specific connection string, then looks if a database mapping is available, finally fallbacks to the `'Default'` connection string.

#### ## Set the Connection String Name

A module typically has a unique connection string name associated to its `'DbContext'` class using the `'ConnectionStringName'` attribute. Example:

```

```csharp
[ConnectionStringName("AbpIdentityServer")]
public class IdentityServerDbContext
    : AbpDbContext<IdentityServerDbContext>, IIIdentityServerDbContext
{
}
```

```

For [\[Entity Framework Core\]](#)(Entity-Framework-Core.md) and [\[MongoDB\]](#)(MongoDB.md), write this to your `'DbContext'` class (and the interface if it has). In this way, ABP uses the specified connection string for the related `'DbContext'` instances.

#### ## Database Migrations for the Entity Framework Core

Relational databases require to create the database and the database schema (tables, views... etc.) before using it.

The startup template (with EF Core ORM) comes with a single database and a `'.EntityFrameworkCore'` project that contains related classes and the migration files for that database. This project mainly defines a `'YourProjectNameDbContext'` class that calls the `'Configure...()'` methods of the used modules, like `'builder.ConfigurePermissionManagement()'`.

Once you want to separate a module's database, you typically will need to create a second migration path. See the [\[EF Core Migrations\]](#)(Entity-

Framework-Core-Migrations.md) document to learn how to create and use a different database for a desired module.

## ## Multi-Tenancy

See [[the multi-tenancy document](#)](Multi-Tenancy.md) to learn how to use separate databases for tenants.

## ## Replace the Connection String Resolver

ABP defines the '[IConnectionStringResolver](#)' and uses it whenever it needs a connection string. It has two pre-built implementations:

- \* '[DefaultConnectionStringResolver](#)' uses the '[AbpDbConnectionOptions](#)' to select the connection string based on the rules defined in the "Configure the Connection Strings" section above.
- \* '[MultiTenantConnectionStringResolver](#)' used for multi-tenant applications and tries to get the configured connection string for the current tenant if available. It uses the '[ITenantStore](#)' to find the connection strings. It inherits from the '[DefaultConnectionStringResolver](#)' and fallbacks to the base logic if no connection string specified for the current tenant.

If you need a custom logic to determine the connection string, implement the '[IConnectionStringResolver](#)' interface (optionally derive from the existing implementations) and replace the existing implementation using the [[dependency injection](#)](Dependency-Injection.md) system.

## 6.6 Dependency Injection

### # Dependency Injection

ABP's Dependency Injection system is developed based on Microsoft's [[dependency injection extension](#)](<https://medium.com/volosoft/asp-net-core-dependency-injection-best-practices-tips-tricks-c6e9c67f9d96>) library (`Microsoft.Extensions.DependencyInjection` nuget package). So, it's documentation is valid in ABP too.

> While ABP has no core dependency to any 3rd-party DI provider. However, it's required to use a provider that supports dynamic proxying and some other advanced features to make some ABP features properly work. Startup templates come with [[Autofac](#)](<https://autofac.org/>) installed. See [[Autofac integration](#)](Autofac-Integration.md) document for more information.

## ## Modularity

Since ABP is a modular framework, every module defines its own services and registers via dependency injection in its own separate [[module class](#)](Module-Development-Basics.md). Example:

```
```C#
public class BlogModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        //register dependencies here
    }
}
```

```
}...
```

Conventional Registration

ABP introduces conventional service registration. You need not do anything to register a service by convention. It's automatically done. If you want to disable it, you can set `'SkipAutoServiceRegistration'` to `'true'` in the constructor of your module class. Example:

```
```C#
public class BlogModule : AbpModule
{
 public BlogModule()
 {
 SkipAutoServiceRegistration = true;
 }
}...
```

Once you skip the auto registration, you should manually register your services. In that case, `'AddAssemblyOf'` extension method can help you to register all your services by convention. Example:

```
```c#
public class BlogModule : AbpModule
{
    public BlogModule()
    {
        SkipAutoServiceRegistration = true;
    }

    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        context.Services.AddAssemblyOf<BlogModule>();
    }
}...
```

The sections below explain the conventions and configurations.

Inherently Registered Types

Some specific types are registered to dependency injection by default. Examples:

- * Module classes are registered as singleton.
- * MVC controllers (inherit `'Controller'` or `'AbpController'`) are registered as transient.
- * MVC page models (inherit `'PageModel'` or `'AbpPageModel'`) are registered as transient.
- * MVC view components (inherit `'ViewComponent'` or `'AbpViewComponent'`) are registered as transient.
- * Application services (inherit `'ApplicationService'` class or its subclasses) are registered as transient.
- * Repositories (implement `'BasicRepositoryBase'` class or its subclasses) are registered as transient.

* Domain services (implement ```IDomainService``` interface or inherit ```DomainService``` class) are registered as transient.

Example:

```
```C#
public class BlogPostAppService : ApplicationService
{
}
...```

```

```BlogPostAppService``` is automatically registered with transient lifetime since it's derived from a known base class.

Dependency Interfaces

If you implement these interfaces, your class is registered to dependency injection automatically:

- * ```ITransientDependency``` to register with transient lifetime.
- * ```ISingletonDependency``` to register with singleton lifetime.
- * ```IScopedDependency``` to register with scoped lifetime.

Example:

```
```C#
public class TaxCalculator : ITransientDependency
{
}
...```

```

```TaxCalculator``` is automatically registered with a transient lifetime since it implements ```ITransientDependency```.

Dependency Attribute

Another way of configuring a service for dependency injection is to use ```DependencyAttribute```. It has the following properties:

- * ```Lifetime```: Lifetime of the registration: ```Singleton```, ```Transient``` or ```Scoped```.
- * ```TryRegister```: Set ```true``` to register the service only if it's not registered before. Uses TryAdd... extension methods of `IServiceCollection`.
- * ```ReplaceServices```: Set ```true``` to replace services if they are already registered before. Uses Replace extension method of `IServiceCollection`.

Example:

```
```C#
[Dependency(ServiceLifetime.Transient, ReplaceServices = true)]
public class TaxCalculator
{
}
...```

```

```Dependency``` attribute has a higher priority than other dependency interfaces if it defines the ```Lifetime``` property.

ExposeServices Attribute

``ExposeServicesAttribute`` is used to control which services are provided by the related class. Example:

```
```C#
[ExposeServices(typeof(ITaxCalculator))]
public class TaxCalculator: ICalculator, ITaxCalculator, ICanCalculate,
ITransientDependency
{
}
...``
```

``TaxCalculator`` class only exposes ``ITaxCalculator`` interface. That means you can only inject ``ITaxCalculator``, but can not inject ``TaxCalculator`` or ``ICalculator`` in your application.

### ### Exposed Services by Convention

If you do not specify which services to expose, ABP expose services by convention. So taking the ``TaxCalculator`` defined above:

- \* The class itself is exposed by default. That means you can inject it by ``TaxCalculator`` class.
- \* Default interfaces are exposed by default. Default interfaces are determined by naming convention. In this example, ``ICalculator`` and ``ITaxCalculator`` are default interfaces of ``TaxCalculator``, but ``ICanCalculate`` is not. A generic interface (e.g. `ICalculator<string>`) is also considered as a default interface if the naming convention is satisfied.

### ### Combining All Together

Combining attributes and interfaces is possible as long as it's meaningful.

```
```C#
[Dependency(ReplaceServices = true)]
[ExposeServices(typeof(ITaxCalculator))]
public class TaxCalculator : ITaxCalculator, ITransientDependency
{
}
...``
```

Manually Registering

In some cases, you may need to register a service to the `IServiceCollection` manually, especially if you need to use custom factory methods or singleton instances. In that case, you can directly add services just as [Microsoft documentation](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection>) describes. Example:

```
```C#
public class BlogModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 //Register an instance as singleton
 }
}```
```

```

 context.Services.AddSingleton<TaxCalculator>(new
TaxCalculator(taxRatio: 0.18));

 //Register a factory method that resolves from IServiceProvider
context.Services.AddScoped<ITaxCalculator>(
 sp => sp.GetRequiredService<TaxCalculator>()
);
}
}...

```

### ### Replace a Service

If you need to replace an existing service (defined by the ABP framework or another module dependency), you have two options;

1. Use the '`Dependency`' attribute of the ABP framework as explained above.
2. Use the '`IServiceCollection.Replace`' method of the Microsoft Dependency Injection library. Example:

```

```csharp
public class MyModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext
context)
    {
        //Replacing the IConnectionStringResolver service
        context.Services.Replace(
            ServiceDescriptor.Transient<
                IConnectionStringResolver,
                MyConnectionStringResolver
            >());
    }
}...

```

Injecting Dependencies

There are three common ways of using a service that has already been registered.

Constructor Injection

This is the most common way of injecting a service into a class. For example:

```

```C#
public class TaxAppService : ApplicationService
{
 private readonly ITaxCalculator _taxCalculator;

 public TaxAppService(ITaxCalculator taxCalculator)
 {
 _taxCalculator = taxCalculator;
 }

 public async Task DoSomethingAsync()
 {
 //...use _taxCalculator...
 }
}

```

```
 }
}
```

``TaxAppService`` gets ``ITaxCalculator`` in it's constructor. The dependency injection system automatically provides the requested service at runtime.

Constructor injection is preferred way of injecting dependencies to a class. In that way, the class can not be constructed unless all constructor-injected dependencies are provided. Thus, the class explicitly declares it's required services.

### ### Property Injection

Property injection is not supported by Microsoft Dependency Injection library. However, ABP can integrate with 3rd-party DI providers ([\[Autofac\]](#)(<https://autofac.org/>), for example) to make property injection possible. Example:

```
```C#
public class MyService : ITransientDependency
{
    public ILogger<MyService> Logger { get; set; }

    public MyService()
    {
        Logger = NullLogger<MyService>.Instance;
    }

    public async Task DoSomethingAsync()
    {
        //...use Logger to write logs...
    }
}...
```

For a property-injection dependency, you declare a public property with public setter. This allows the DI framework to set it after creating your class.

Property injected dependencies are generally considered as **optional** dependencies. That means the service can properly work without them. ``Logger`` is such a dependency, ``MyService`` can continue to work without logging.

To make the dependency properly optional, we generally set a default/fallback value to the dependency. In this sample, NullLogger is used as fallback. Thus, ``MyService`` can work but does not write logs if DI framework or you don't set Logger property after creating ``MyService``.

One restriction of property injection is that you cannot use the dependency in your constructor, since it's set after the object construction.

Property injection is also useful when you want to design a base class that has some common services injected by default. If you're going to use constructor injection, all derived classes should also inject depended services into their own constructors which makes development harder. However,

be very careful using property injection for non-optional services as it makes it harder to clearly see the requirements of a class.

DisablePropertyInjection Attribute

You can use ` [DisablePropertyInjection]` attribute on classes or their properties to disable property injection for the whole class or some specific properties.

```
```C#
// Disabling for all properties of the MyService class
[DisablePropertyInjection]
public class MyService : ITransientDependency
{
 public ILogger<MyService> Logger { get; set; }

 public ITaxCalculator TaxCalculator { get; set; }
}

// Disabling only for the TaxCalculator property
public class MyService : ITransientDependency
{
 public ILogger<MyService> Logger { get; set; }

 [DisablePropertyInjection]
 public ITaxCalculator TaxCalculator { get; set; }
}
````
```

Resolve Service from IServiceProvider

You may want to resolve a service directly from ``IServiceProvider``. In that case, you can inject `IServiceProvider` into your class and use the ``GetService`` or the `GetRequiredService` method as shown below:

```
```C#
public class MyService : ITransientDependency
{
 private readonly ITaxCalculator _taxCalculator;

 public MyService(IServiceProvider serviceProvider)
 {
 _taxCalculator =
serviceProvider.GetRequiredService<ITaxCalculator>();
 }
}
````
```

Dealing with multiple implementations

You can register multiple implementations of the same service interface. Assume that you have an `IExternalLogger` interface with two implementations:

```
```csharp
public interface IExternalLogger
{
 Task LogAsync(string logText);
}
```

```

public class ElasticsearchExternalLogger : IExternalLogger
{
 public async Task LogAsync(string logText)
 {
 //TODO...
 }
}

public class AzureExternalLogger : IExternalLogger
{
 public Task LogAsync(string logText)
 {
 throw new System.NotImplementedException();
 }
}
...

```

In this example, we haven't registered any of the implementation classes to the dependency injection system yet. So, if we try to inject the `IExternalLogger` interface, we get an error indicating that no implementation found.

If we register both of the `ElasticsearchExternalLogger` and `AzureExternalLogger` services for the `IExternalLogger` interface, and then try to inject the `IExternalLogger` interface, then the last registered implementation will be used.

An example service injecting the `IExternalLogger` interface:

```

```csharp
public class MyService : ITransientDependency
{
    private readonly IExternalLogger _externalLogger;

    public MyService(IExternalLogger externalLogger)
    {
        _externalLogger = externalLogger;
    }

    public async Task DemoAsync()
    {
        await _externalLogger.LogAsync("Example log message...");
    }
}
...

```

Here, as said before, we get the last registered implementation. However, how to determine the last registered implementation?

If we implement one of the dependency interfaces (e.g. `ITransientDependency`), then the registration order will be uncertain (it may depend on the namespaces of the classes). The **last registered implementation** can be different than you expect. So, it is not suggested to use the dependency interfaces to register multiple implementations.

You can register your services in the `ConfigureServices` method of your module:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 context.Services.AddTransient<IExternalLogger,
ElasticsearchExternalLogger>();
 context.Services.AddTransient<IExternalLogger, AzureExternalLogger>();
}
```

```

In this case, you get an `AzureExternalLogger` instance when you inject the `IExternalLogger` interface, because the last registered implementation is the `AzureExternalLogger` class.

When you have multiple implementation of an interface, you may want to work with all these implementations. Assume that you want to write log to all the external loggers. We can change the `MyService` implementation as the following:

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IEnumerable<IExternalLogger> _externalLoggers;

 public MyService(IEnumerable<IExternalLogger> externalLoggers)
 {
 _externalLoggers = externalLoggers;
 }

 public async Task DemoAsync()
 {
 foreach (var externalLogger in _externalLoggers)
 {
 await externalLogger.LogAsync("Example log message...");
 }
 }
}
```

```

In this example, we are injecting `IEnumerable<IExternalLogger>` instead of `IExternalLogger`, so we have a collection of the `IExternalLogger` implementations. Then we are using a `foreach` loop to write the same log text to all the `IExternalLogger` implementations.

If you are using `IServiceProvider` to resolve dependencies, then use its `GetServices` method to obtain a collection of the service implementations:

```
```csharp
IEnumerable<IExternalLogger> services =
_serviceProvider.GetServices<IExternalLogger>();
```

```

Releasing/Disposing Services

If you used a constructor or property injection, you don't need to be concerned about releasing the service's resources. However, if you have resolved a service from `IServiceProvider`, in some cases, you might need to take care about releasing the service resources.

ASP.NET Core releases all services at the end of a current HTTP request, even if you directly resolved from ```IServiceProvider``` (assuming you injected `IServiceProvider`). But, there are several cases where you may want to release/dispose manually resolved services:

- * Your code is executed outside of ASP.NET Core request and the executer hasn't handled the service scope.
- * You only have a reference to the root service provider.
- * You may want to immediately release & dispose services (for example, you may creating too many services with big memory usages and don't want to overuse the memory).

In any case, you can create a service scope block to safely and immediately release services:

```
```C#
using (var scope = _serviceProvider.CreateScope())
{
 var service1 = scope.ServiceProvider.GetService<IMyService1>();
 var service2 = scope.ServiceProvider.GetService<IMyService2>();
}
...```

```

Both services are released when the created scope is disposed (at the end of the `'using'` block).

### ### Cached Service Providers

ABP provides two special services to optimize resolving services from `IServiceProvider`. `ICachedServiceProvider` and `ITransientCachedServiceProvider` both inherits from the `IServiceProvider` interface and internally caches the resolved services, so you get the same service instance even if you resolve a service multiple times.

The main difference is the `ICachedServiceProvider` is itself registered as scoped, while the `ITransientCachedServiceProvider` is registered as transient to the dependency injection system.

The following example injects the `ICachedServiceProvider` service and resolves a service in the `DoSomethingAsync` method:

```
```csharp
public class MyService : ITransientDependency
{
    private readonly ICachedServiceProvider _serviceProvider;

    public MyService(ICachedServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    public async Task DoSomethingAsync()
    {
        var taxCalculator =
(serviceProvider.GetRequiredService<ITaxCalculator>());
        // TODO: Use the taxCalculator
    }
}
```

```

3

1

With such a usage, you don't need to deal with creating service scopes and disposing the resolved services (as explained in the *\*Releasing/Disposing Services\** section above). Because all the services resolved from the `'ICachedServiceProvider'` will be released once the service scope of the `'MyService'` instance is disposed. Also, you don't need to care about memory leaks (because of creating too many `'ITaxCalculator'` instances if we call `'DoSomethingAsync'` too many times), because only one `'ITaxCalculator'` instance is created, and it is reused.

Since `'ICachedServiceProvider'` and `'ITransientCachedServiceProvider'` extends the standard `'IServiceProvider'` interface, you can use all the extension method of the `'IServiceProvider'` interface on them. In addition, they provides some other methods to provide a default value or a factory method for the services that are not found (that means not registered to the dependency injection system). Notice that the default value (or the value returned from your factory method) is also cached and reused.

Use `'ICachedServiceProvider'` (instead of `'ITransientCachedServiceProvider'`) unless you need to create the service cache per usage.

`'ITransientCachedServiceProvider'` guarantees that the created service instances are not shared with any other service, even they are in the same service scope. The services resolved from `'ICachedServiceProvider'` are shared with other services in the same service scope (in the same HTTP Request, for example), so it can be thought as more optimized.

> ABP Framework also provides the `IAbpLazyServiceProvider` service. It does exist for backward compatibility and works exactly same with the `ITransientCachedServiceProvider` service. So, use the `ITransientCachedServiceProvider` since the `IAbpLazyServiceProvider` might be removed in future ABP versions.

## ## Advanced Features

### ### IServiceCollection.OnRegistered Event

You may want to perform an action for every service registered to the dependency injection. In the '`PreConfigureServices`' method of your module, register a callback using the '`OnRegistered`' method as shown below:

```
```csharp
public class AppModule : AbpModule
{
    public override void PreConfigureServices(ServiceConfigurationContext context)
    {
        context.Services.OnRegistered(ctx =>
        {
            var type = ctx.ImplementationType;
            //...
        });
    }
}
```
```

`ImplementationType` provides the service type. This callback is generally used to add interceptor to a service. Example:

```
```csharp
public class AppModule : AbpModule
{
    public override void PreConfigureServices(ServiceConfigurationContext context)
    {
        context.Services.OnRegistered(ctx =>
        {
            if (ctx.ImplementationType.IsDefined(typeof(MyLogAttribute), true))
            {
                ctx.Interceptors.TryAdd<MyLogInterceptor>();
            }
        });
    }
}...```

```

This example simply checks if the service class has `MyLogAttribute` attribute and adds `MyLogInterceptor` to the interceptor list if so.

> Notice that `OnRegistered` callback might be called multiple times for the same service class if it exposes more than one service/interface. So, it's safe to use `Interceptors.TryAdd` method instead of `Interceptors.Add` method. See [[the documentation](#)](Dynamic-Proxying-Interceptors.md) of dynamic proxying / interceptors.

3rd-Party Providers

While ABP has no core dependency to any 3rd-party DI provider, it's required to use a provider that supports dynamic proxying and some other advanced features to make some ABP features properly work.

Startup templates come with Autofac installed. See [[Autofac integration](#)](Autofac-Integration.md) document for more information.

See Also

- * [[ASP.NET Core Dependency Injection Best Practices, Tips & Tricks](#)](<https://medium.com/volosoft/asp-net-core-dependency-injection-best-practices-tips-tricks-c6e9c67f9d96>)

6.6.1 AutoFac Integration

Autofac Integration

[[Autofac](#)](<https://autofac.org/>) is one of the most used dependency injection frameworks for .NET. It provides advanced features compared to .Net Core's standard DI library, like dynamic proxying and property injection.

Install Autofac Integration

> All the [startup templates](Startup-Templates/Index.md) and samples are Autofac integrated. So, most of the time you don't need to manually install this package.

If you're not using a startup template, you can use the [ABP CLI](CLI.md) to install it to your project. Execute the following command in the folder that contains the .csproj file of your project (suggested to add it to the executable/web project):

```
```bash
abp add-package Volo.Abp.Autofac
````
```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.Autofac).

>

Finally, configure `AbpApplicationCreationOptions` to replace default dependency injection services by Autofac. It depends on the application type.

ASP.NET Core Application

Call `UseAutofac()` in the **Program.cs** file as shown below:

```
```csharp
public class Program
{
 public static int Main(string[] args)
 {
 CreateHostBuilder(args).Build().Run();
 }

 internal static IHostBuilder CreateHostBuilder(string[] args) =>
 Host.CreateDefaultBuilder(args)
 .ConfigureWebHostDefaults(webBuilder =>
 {
 webBuilder.UseStartup<Startup>();
 })
 .UseAutofac(); //Integrate Autofac!
}
````
```

If you are using the static `WebApplication` class, you can call the `UseAutofac()` extension method as shown below:

```
```csharp
public class Program
{
 public async static Task Main(string[] args)
 {
 var builder = WebApplication.CreateBuilder(args);
 builder.Host.UseAutofac(); // Integrate Autofac!
 await builder.AddApplicationAsync<MyProjectNameWebModule>();
 var app = builder.Build();
 await app.InitializeApplicationAsync();
 await app.RunAsync();
 }
}
````
```

```

} ...

### Console Application

Call `UseAutofac()` method in the `AbpApplicationFactory.Create` options as
shown below:

```csharp
using System;
using Microsoft.Extensions.DependencyInjection;
using Volo.Abp;

namespace AbpConsoleDemo
{
 class Program
 {
 static void Main(string[] args)
 {
 using (var application =
AbpApplicationFactory.Create<AppModule>(options =>
{
 options.UseAutofac(); //Autofac integration
})) {
 //...
 }
 }
}
```

```

Using the Autofac Registration API

If you want to use Autofac's advanced [registration API](<https://autofac.readthedocs.io/en/latest/register/registration.html>), you need to access the `ContainerBuilder` object. [Volo.Abp.Autofac](<https://www.nuget.org/packages/Volo.Abp.Autofac>) nuget package defines the `IServiceCollection.GetContainerBuilder()` extension method to obtain the `ContainerBuilder` object.

****Example: Get the `ContainerBuilder` object in the `ConfigureServices` method of your [module class](Module-Development-Basics.md)****

```

```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 var containerBuilder = context.Services.GetContainerBuilder();
 containerBuilder.RegisterType<MyService>(); // Using Autofac's
registration API
}
```

```

> You should install the [Volo.Abp.Autofac](<https://www.nuget.org/packages/Volo.Abp.Autofac>) nuget package to the project that you want to use the Autofac API.

6.7 Exception Handling

Exception Handling

ABP provides a built-in infrastructure and offers a standard model for handling exceptions.

- * Automatically **handles all exceptions** and sends a standard **formatted error message** to the client for an API/AJAX request.
- * Automatically hides **internal infrastructure errors** and returns a standard error message.
- * Provides an easy and configurable way to **localize** exception messages.
- * Automatically maps standard exceptions to **HTTP status codes** and provides a configurable option to map custom exceptions.

Automatic Exception Handling

'`AbpExceptionFilter`' handles an exception if **any of the following conditions** are met:

- * Exception is thrown by a **controller action** which returns an **object result** (not a view result).
- * The request is an AJAX request ('`X-Requested-With`' HTTP header value is '`XMLHttpRequest`').
- * Client explicitly accepts the '`application/json`' content type (via '`accept`' HTTP header).

If the exception is handled it's automatically **logged** and a formatted **JSON message** is returned to the client.

Error Message Format

Error Message is an instance of the '`RemoteServiceErrorResponse`' class. The simplest error JSON has a **message** property as shown below:

```
```json
{
 "error": {
 "message": "This topic is locked and can not add a new message"
 }
}...```

```

There are \*\*optional fields\*\* those can be filled based upon the exception that has occurred.

### ##### Error Code

Error \*\*code\*\* is an optional and unique string value for the exception. Thrown '`Exception`' should implement the '`IHasErrorCode`' interface to fill this field. Example JSON value:

```
```json
{
  "error": {
    "code": "App:010042",
    "message": "This topic is locked and can not add a new message"
  }
}...```

```

```
    }
}
...  
}
```

Error code can also be used to localize the exception and customize the HTTP status code (see the related sections below).

Error Details

Error ****details**** in an optional field of the JSON error message. Thrown `'Exception'` should implement the `'IHasErrorDetails'` interface to fill this field. Example JSON value:

```
```json
{
 "error": {
 "code": "App:010042",
 "message": "This topic is locked and can not add a new message",
 "details": "A more detailed info about the error..."
 }
}
...
```
}
```

Validation Errors

****validationErrors**** is a standard field that is filled if the thrown exception implements the `'IHasValidationErrors'` interface.

```
```json
{
 "error": {
 "code": "App:010046",
 "message": "Your request is not valid, please correct and try again!",
 "validationErrors": [
 {
 "message": "Username should be minimum length of 3.",
 "members": ["userName"]
 },
 {
 "message": "Password is required",
 "members": ["password"]
 }
]
}
...
```
}
```

`'AbpValidationException'` implements the `'IHasValidationErrors'` interface and it is automatically thrown by the framework when a request input is not valid. So, usually you don't need to deal with validation errors unless you have highly customised validation logic.

Logging

Caught exceptions are automatically logged.

Log Level

Exceptions are logged with the 'Error' level by default. The Log level can be determined by the exception if it implements the 'IHasLogLevel' interface.
Example:

```
```C#
public class MyException : Exception, IHasLogLevel
{
 public LogLevel LogLevel { get; set; } = LogLevel.Warning;

 //...
}```
```

#### #### Self Logging Exceptions

Some exception types may need to write additional logs. They can implement the 'IExceptionWithSelfLogging' if needed. Example:

```
```C#
public class MyException : Exception, IExceptionWithSelfLogging
{
    public void Log(ILogger logger)
    {
        //...log additional info
    }
}```
```

> 'ILogger.LogError' extension methods is used to write exception logs.
You can use the same extension method when needed.

Business Exceptions

Most of your own exceptions will be business exceptions. The 'IBusinessException' interface is used to mark an exception as a business exception.

'BusinessException' implements the 'IBusinessException' interface in addition to the 'IHasErrorCode', 'IHasErrorDetails' and 'IHasLogLevel' interfaces. The default log level is 'Warning'.

Usually you have an error code related to a particular business exception.
For example:

```
```C#
throw new BusinessException(QaErrorCodes.CanNotVoteYourOwnAnswer);
```
```

'QaErrorCodes.CanNotVoteYourOwnAnswer' is just a 'const string'. The following error code format is recommended:

```
```
<code-namespace>:<error-code>
```
```

code-namespace is a **unique value** specific to your module/application.
Example:

```
....
```

```
Volo.Qa:010002
```

```
`````  
'Volo.Qa' is the code-namespace here. code-namespace is then will be used
while **localizing** exception messages.
```

- \* You can \*\*directly throw\*\* a '[BusinessException](#)' or \*\*derive\*\* your own exception types from it when needed.
- \* All properties are optional for the '[BusinessException](#)' class. But you generally set either '[ErrorCode](#)' or '[Message](#)' property.

## ## Exception Localization

One problem with throwing exceptions is how to localize error messages while sending it to the client. ABP offers two models and their variants.

### ### User Friendly Exception

If an exception implements the '[IUserFriendlyException](#)' interface, then ABP does not change it's '[Message](#)' and '[Details](#)' properties and directly send it to the client.

'[UserFriendlyException](#)' class is the built-in implementation of the '[IUserFriendlyException](#)' interface. Example usage:

```
```C#  
throw new UserFriendlyException(  
    "Username should be unique!"  
);  
....
```

In this way, there is **no need for localization** at all. If you want to localize the message, you can inject and use the standard **string localizer** (see the [[localization document](#)](Localization.md)). Example:

```
```C#  
throw new
UserFriendlyException(_stringLocalizer["UserNameShouldBeUniqueMessage"]);
....
```

Then define it in the \*\*localization resource\*\* for each language. Example:

```
```json  
{  
    "culture": "en",  
    "texts": {  
        "UserNameShouldBeUniqueMessage": "Username should be unique!"  
    }  
};  
....
```

String localizer already supports **parameterized messages**. For example:

```
```C#  
throw new
UserFriendlyException(_stringLocalizer["UserNameShouldBeUniqueMessage",
 "john"]);
....
```

....

Then the localization text can be:

```
```json
"UserNameShouldBeUniqueMessage": "Username should be unique! '{0}' is already
taken!"
````
```

\* The `IUserFriendlyException` interface is derived from the `IBusinessException` and the `UserFriendlyException` class is derived from the `BusinessException` class.

### ### Using Error Codes

`UserFriendlyException` is fine, but it has a few problems in advanced usages:

- \* It requires you to \*\*inject the string localizer\*\* everywhere and always use it while throwing exceptions.
- \* However, in some of the cases, it may \*\*not be possible\*\* to inject the string localizer (in a static context or in an entity method).

Instead of localizing the message while throwing the exception, you can separate the process using \*\*error codes\*\*.

First, define the \*\*code-namespace\*\* to \*\*localization resource\*\* mapping in the module configuration:

```
```C#
services.Configure<AbpExceptionLocalizationOptions>(options =>
{
    options.MapCodeNamespace("Volo.Qa", typeof(QaResource));
});
````
```

Then any of the exceptions with `Volo.Qa` namespace will be localized using their given localization resource. The localization resource should always have an entry with the error code key. Example:

```
```json
{
    "culture": "en",
    "texts": {
        "Volo.Qa:010002": "You can not vote your own answer!"
    }
}
````
```

Then a business exception can be thrown with the error code:

```
```C#
throw new BusinessException(QaDomainErrorCodes.CanNotVoteYourOwnAnswer);
````
```

\* Throwing any exception implementing the `IHasErrorCode` interface behaves the same. So, the error code localization approach is not unique to the `BusinessException` class.

\* Defining localized string is not required for an error message. If it's not defined, ABP sends the default error message to the client. It does not use the `Message` property of the exception! if you want that, use the `UserFriendlyException` (or use an exception type that implements the `IUserFriendlyException` interface).

#### #### Using Message Parameters

If you have a parameterized error message, then you can set it with the exception's `Data` property. For example:

```
```C#
throw new BusinessException("App:010046")
{
    Data =
    {
        {"UserName", "john"}
    }
};

...```

```

Fortunately there is a shortcut way to code this:

```
```C#
throw new BusinessException("App:010046")
 .WithData("UserName", "john");
...```

```

Then the localized text can contain the `UserName` parameter:

```
```json
{
    "culture": "en",
    "texts": {
        "App:010046": "Username should be unique. '{UserName}' is already taken!"
    }
}...```

```

* `WithData` can be chained with more than one parameter (like `WithData(...).WithData(...)`) .

HTTP Status Code Mapping

ABP tries to automatically determine the most suitable HTTP status code for common exception types by following these rules:

- * For the `AbpAuthorizationException`:
 - * Returns `401` (unauthorized) if user has not logged in.
 - * Returns `403` (forbidden) if user has logged in.
- * Returns `400` (bad request) for the `AbpValidationException`.
- * Returns `404` (not found) for the `EntityNotFoundException`.
- * Returns `403` (forbidden) for the `IBusinessException` (and `IUserFriendlyException` since it extends the `IBusinessException`).
- * Returns `501` (not implemented) for the `NotImplementedException`.
- * Returns `500` (internal server error) for other exceptions (those are assumed as infrastructure exceptions).

The '`IHttpExceptionStatusCodeFinder`' is used to automatically determine the HTTP status code. The default implementation is the '`DefaultHttpExceptionStatusCodeFinder`' class. It can be replaced or extended as needed.

Custom Mappings

Automatic HTTP status code determination can be overrided by custom mappings. For example:

```
```C#
services.Configure<AbpExceptionHttpStatusCodeOptions>(options =>
{
 options.Map("Volo.Qa:010002", HttpStatusCode.Conflict);
});```

```

### ## Subscribing to the Exceptions

It is possible to be informed when the ABP Framework \*\*handles\*\* an exception\*\*. It automatically \*\*logs\*\* all the exceptions to the standard [logger](Logging.md), but you may want to do more.

In this case, create a class derived from the '`ExceptionSubscriber`' class in your application:

```
```csharp
public class MyExceptionSubscriber : ExceptionSubscriber
{
    public async override Task HandleAsync(ExceptionNotificationContext context)
    {
        //TODO...
    }
}```

```

The '`context`' object contains necessary information about the exception occurred.

> You can have multiple subscribers, each gets a copy of the exception. Exceptions thrown by your subscriber is ignored (but still logged).

Built-In Exceptions

Some exception types are automatically thrown by the framework:

- '`AbpAuthorizationException`' is thrown if the current user has no permission to perform the requested operation. See [authorization](Authorization.md) for more.
- '`AbpValidationException`' is thrown if the input of the current request is not valid. See [validation](Validation.md) for more.
- '`EntityNotFoundException`' is thrown if the requested entity is not available. This is mostly thrown by [repositories](Repositories.md).

You can also throw these type of exceptions in your code (although it's rarely needed).

```

## AbpExceptionHandlingOptions

`'AbpExceptionHandlingOptions` is the main [options object](Options.md) to
configure the exception handling system. You can configure it in the
`ConfigureServices` method of your [module](Module-Development-Basics.md):

```csharp
Configure<AbpExceptionHandlingOptions>(options =>
{
 options.SendExceptionsDetailsToClients = true;
 options.SendStackTraceToClients = false;
});
```

```

Here, a list of the options you can configure:

- * `SendExceptionsDetailsToClients` (default: `false`): You can enable or disable sending exception details to the client.
- * `SendStackTraceToClients` (default: `true`): You can enable or disable sending the stack trace of exception to the client. If you want to send the stack trace to the client, you must set both `SendStackTraceToClients` and `SendExceptionsDetailsToClients` options to `true` otherwise, the stack trace will not be sent to the client.

6.8 Localization

Localization

ABP's localization system is seamlessly integrated to the `'Microsoft.Extensions.Localization'` package and compatible with the [Microsoft's localization documentation](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization>). It adds some useful features and enhancements to make it easier to use in real life application scenarios.

Installation

> This package is already installed by default with the startup template. So, most of the time, you don't need to install it manually.

You can use the [ABP CLI](CLI.md) to install the `Volo.Abp.Localization` package to your project. Execute the following command in the folder of the `.csproj` file that you want to install the package on:

```

```bash
abp add-package Volo.Abp.Localization
```

```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](<https://abp.io/package-detail/Volo.Abp.Localization>).

Then you can add `**AbpLocalizationModule**` dependency to your module:

```

```c#

```

```

using Volo.Abp.Modularity;
using Volo.Abp.Localization;

namespace MyCompany.MyProject
{
 [DependsOn(typeof(AbpLocalizationModule))]
 public class MyModule : AbpModule
 {
 //...
 }
}
```

```

Creating A Localization Resource

A localization resource is used to group related localization strings together and separate them from other localization strings of the application. A `[module](Module-Development-Basics.md)` generally defines its own localization resource. Localization resource is just a plain class. Example:

```

```C#
public class TestResource
{
}
```

```

Then it should be added using `'AbpLocalizationOptions'` as shown below:

```

```C#
[DependsOn(typeof(AbpLocalizationModule))]
public class MyModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 Configure<AbpVirtualFileSystemOptions>(options =>
 {
 // "YourRootNameSpace" is the root namespace of your project. It can be empty if your root namespace is empty.
 options.FileSets.AddEmbedded<MyModule>("YourRootNameSpace");
 });

 Configure<AbpLocalizationOptions>(options =>
 {
 //Define a new localization resource (TestResource)
 options.Resources
 .Add<TestResource>("en")
 .AddVirtualJson("/Localization/Resources/Test");
 });
 }
}
```

```

In this example;

* Added a new localization resource with "en" (English) as the default culture.

```
* Used JSON files to store the localization strings.  
* JSON files are embedded into the assembly using  
'AbpVirtualFileSystemOptions' (see [virtual file system](Virtual-File-System.md)).
```

JSON files are located under "/Localization/Resources/Test" project folder as shown below:

```
![localization-resource-json-files](images/localization-resource-json-files.png)
```

A JSON localization file content is shown below:

```
```json  
{
 "culture": "en",
 "texts": {
 "HelloWorld": "Hello World!"
 }
}...
...
```

```
* Every localization file should define the 'culture' code for the file (like "en" or "en-US").
* 'texts' section just contains key-value collection of the localization strings (keys may have spaces too).
```

> ABP will ignore (skip) the JSON file if the '`culture`' section is missing.

### ### Default Resource

'`AbpLocalizationOptions.DefaultResourceType`' can be set to a resource type, so it is used when the localization resource was not specified:

```
```csharp  
Configure<AbpLocalizationOptions>(options =>  
{  
    options.DefaultResourceType = typeof(TestResource);  
});  
...  
...
```

> The [application startup template](Startup-Templates/Application.md) sets '`DefaultResourceType`' to the localization resource of the application.

Short Localization Resource Name

Localization resources are also available in the client (JavaScript) side. So, setting a short name for the localization resource makes it easy to use localization texts. Example:

```
```C#  
[LocalizationResourceName("Test")]
public class TestResource
{
}
...
...
```

See the Getting Localized Test / Client Side section below.

### ### Inherit From Other Resources

A resource can inherit from other resources which makes possible to re-use existing localization strings without referring the existing resource.  
Example:

```
```C#
[InheritResource(typeof(AbpValidationResource))]
public class TestResource
{
}
```
```

Alternative inheritance by configuring the `AbpLocalizationOptions`:

```
```C#
services.Configure<AbpLocalizationOptions>(options =>
{
    options.Resources
        .Add<TestResource>("en") //Define the resource by "en" default
culture
        .AddVirtualJson("/Localization/Resources/Test") //Add strings from
virtual json files
        .AddBaseTypes(typeof(AbpValidationResource)); //Inherit from an
existing resource
});
```
```

\* A resource may inherit from multiple resources.  
\* If the new resource defines the same localized string, it overrides the string.

### ### Extending Existing Resource

Inheriting from a resource creates a new resource without modifying the existing one. In some cases, you may want to not create a new resource but directly extend an existing resource. Example:

```
```C#
services.Configure<AbpLocalizationOptions>(options =>
{
    options.Resources
        .Get<TestResource>()
        .AddVirtualJson("/Localization/Resources/Test/Extensions");
});
```
```

\* If an extension file defines the same localized string, it overrides the string.

### ## Getting the Localized Texts

Getting the localized text is pretty standard.

### ### Simplest Usage In A Class

Just inject the `IStringLocalizer<TResource>` service and use it like shown below:

```
```csharp
public class MyService : ITransientDependency
{
    private readonly IStringLocalizer<TestResource> _localizer;

    public MyService(IStringLocalizer<TestResource> localizer)
    {
        _localizer = localizer;
    }

    public void Foo()
    {
        var str = _localizer["HelloWorld"];
    }
}...```

```

Format Arguments

Format arguments can be passed after the localization key. If your message is `Hello {0}, welcome!`, then you can pass the `{0}` argument to the localizer like `_localizer["HelloMessage", "John"]`.

> Refer to the [Microsoft's localization documentation](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization>) for details about using the localization.

Using In A Razor View/Page

Use `IHtmlLocalizer<T>` in razor views/pages;

```
```c#
@inject IHtmlLocalizer<TestResource> Localizer

<h1>@Localizer["HelloWorld"]</h1>
```

```

Special Base Classes

Some ABP Framework base classes provide a `L` property to use the localizer even easier.

Example: Localize a text in an application service method

```
```csharp
using System.Threading.Tasks;
using MyProject.Localization;
using Volo.Abp.Application.Services;

namespace MyProject
{
 public class TestAppService : ApplicationService
 {
 public TestAppService()
 {
 var str = L["HelloWorld"];
 }
}
```

```

```

    {
        LocalizationResource = typeof(MyProjectResource);
    }

    public async Task DoIt()
    {
        var str = L["HelloWorld"];
    }
}
...

```

When you set the `LocalizationResource` in the constructor, the `ApplicationService` class uses that resource type when you use the `L` property, just like in the `DoIt()` method.

Setting `LocalizationResource` in every application service can be tedious. You can create an abstract base application service class, set it there and derive your application services from that base class. This is already implemented when you create a new project with the [startup templates](Startup-Templates/Application.md). So, you can simply inherit from the base class directly use the `L` property:

```

```csharp
using System.Threading.Tasks;

namespace MyProject
{
 public class TestAppService : MyProjectAppService
 {
 public async Task DoIt()
 {
 var str = L["HelloWorld"];
 }
 }
}
```

```

The `L` property is also available for some other base classes like `AbpController` and `AbpPageModel`.

The Client Side

See the following documents to learn how to reuse the same localization texts in the JavaScript side;

- * [Localization for the MVC / Razor Pages UI](UI/AspNetCore/JavaScript-API/Localization.md)
- * [Localization for the Blazor UI](UI/Blazor/Localization.md)
- * [Localization for the Angular UI](UI/Angular/Localization.md)

6.9 Logging

Logging

ABP Framework doesn't implement any logging infrastructure. It uses the [ASP.NET Core's logging system](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/logging>).

> .NET Core's logging system is actually independent from the ASP.NET Core. It is usable in any type of application.

6.10 Object Extensions

Object Extensions

ABP Framework provides an **object extension system** to allow you to **add extra properties** to an existing object **without modifying** the related class. This allows to extend functionalities implemented by a depended [application module](Modules/Index.md), especially when you want to [extend entities](Customizing-Application-Modules-Extending-Entities.md) and [DTOs](Customizing-Application-Modules-Overriding-Services.md) defined by the module.

> Object extension system normally is not needed for your own objects since you can easily add regular properties to your own classes.

IHasExtraProperties Interface

This is the interface to make a class extensible. It simply defines a `Dictionary` property:

```
```csharp
ExtraPropertyDictionary ExtraProperties { get; }
````
```

`ExtraPropertyDictionary` class is inherited from the `Dictionary<string, object>` class. You can add or get extra properties using this dictionary.

Base Classes

`IHasExtraProperties` interface is implemented by several base classes by default:

- * Implemented by the `AggregateRoot` class (see [entities](Entities.md)).
- * Implemented by `ExtensibleEntityDto`, `ExtensibleAuditedEntityDto`... base [DTO](Data-Transfer-Objects.md) classes.
- * Implemented by the `ExtensibleObject`, which is a simple base class can be inherited for any type of object.

So, if you inherit from these classes, your class will also be extensible. If not, you can always implement it manually.

Fundamental Extension Methods

While you can directly use the `ExtraProperties` property of a class, it is suggested to use the following extension methods while working with the extra properties.

SetProperty

Used to set the value of an extra property:

```
```csharp
user SetProperty("Title", "My Title");
user SetProperty("IsSuperUser", true);
```
```

'SetProperty' returns the same object, so you can chain it:

```
```csharp
user SetProperty("Title", "My Title")
 .SetProperty("IsSuperUser", true);
```
```

GetProperty

Used to read the value of an extra property:

```
```csharp
var title = user.GetProperty<string>("Title");

if (user.GetProperty<bool>("IsSuperUser"))
{
 //...
}
````
```

* 'GetProperty' is a generic method and takes the object type as the generic parameter.
* Returns the default value if given property was not set before (default value is '0' for 'int', 'false' for 'bool'... etc).

Non Primitive Property Types

If your property type is not a primitive (int, bool, enum, string... etc) type, then you need to use non-generic version of the 'GetProperty' which returns an 'object'.

HasProperty

Used to check if the object has a property set before.

RemoveProperty

Used to remove a property from the object. Use this methods instead of setting a 'null' value for the property.

Some Best Practices

Using magic strings for the property names is dangerous since you can easily type the property name wrong - it is not type safe. Instead;

- * Define a constant for your extra property names
- * Create extension methods to easily set your extra properties.

Example:

```
```csharp
public static class IdentityUserExtensions
{
 private const string TitlePropertyName = "Title";

 public static void SetTitle(this IdentityUser user, string title)
 {
 user SetProperty(TitlePropertyName, title);
 }

 public static string GetTitle(this IdentityUser user)
 {
 return user.GetProperty<string>(TitlePropertyName);
 }
}
```

```

Then you can easily set or get the 'Title' property:

```
```csharp
user.SetTitle("My Title");
var title = user.GetTitle();
```

```

Object Extension Manager

While you can set arbitrary properties to an extensible object (which implements the 'IHasExtraProperties' interface), 'ObjectExtensionManager' is used to explicitly define extra properties for extensible classes.

Explicitly defining an extra property has some use cases:

- * Allows to control how the extra property is handled on object to object mapping (see the section below).
- * Allows to define metadata for the property. For example, you can map an extra property to a table field in the database while using the [EF Core](Entity-Framework-Core.md).

> 'ObjectExtensionManager' implements the singleton pattern ('ObjectExtensionManager.Instance') and you should define object extensions before your application startup. The [application startup template](Startup-Templates/Application.md) has some pre-defined static classes to safely define object extensions inside.

AddOrUpdate

'AddOrUpdate' is the main method to define extra properties or update extra properties for an object.

Example: Define extra properties for the 'IdentityUser' entity:

```
```csharp
ObjectExtensionManager.Instance
 .AddOrUpdate<IdentityUser>(options =>
{
 options.AddOrUpdateProperty<string>("SocialSecurityNumber");
 options.AddOrUpdateProperty<bool>("IsSuperUser");
})
```

```

```
    ...);  
    ...
```

AddOrUpdateProperty

While `'AddOrUpdateProperty'` can be used on the `'options'` as shown before, if you want to define a single extra property, you can use the shortcut extension method too:

```
```csharp  
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<IdentityUser, string>("SocialSecurityNumber");
```
```

Sometimes it would be practical to define a single extra property to multiple types. Instead of defining one by one, you can use the following code:

```
```csharp  
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<string>(
 new[]
 {
 typeof(IdentityUserDto),
 typeof(IdentityUserCreateDto),
 typeof(IdentityUserUpdateDto)
 },
 "SocialSecurityNumber"
);
```
```

Property Configuration

`'AddOrUpdateProperty'` can also get an action that can perform additional configuration on the property definition:

```
```csharp  
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<IdentityUser, string>(
 "SocialSecurityNumber",
 options =>
 {
 //Configure options...
 });
```
```

> `'options'` has a dictionary, named `'Configuration'` which makes the object extension definitions even extensible. It is used by the EF Core to map extra properties to table fields in the database. See the [\[extending entities\]\(Customizing-Application-Modules-Extending-Entities.md\)](#) document.

The following sections explain the fundamental property configuration options.

Default Value

A default value is automatically set for the new property, which is the natural default value for the property type, like `'null'` for `'string'`, `'false'` for `'bool'` or `'0'` for `'int'`.

There are two ways to override the default value:

DefaultValue Option

'DefaultValue' option can be set to any value:

```
```csharp
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<IdentityUser, int>(
 "MyIntProperty",
 options =>
 {
 options.DefaultValue = 42;
 });
...```

```

#### ##### DefaultValueFactory Options

'DefaultValueFactory' can be set to a function that returns the default value:

```
```csharp
ObjectExtensionManager.Instance
    .AddOrUpdateProperty<IdentityUser, DateTime>(
        "MyDateTimeProperty",
        options =>
    {
        options.DefaultValueFactory = () => DateTime.Now;
    });
...```

```

'options.DefaultValueFactory' has a higher priority than the 'options.DefaultValue'.

> Tip: Use 'DefaultValueFactory' option only if the default value may change over the time (like 'DateTime.Now' in this example). If it is a constant value, then use the 'DefaultValue' option.

CheckPairDefinitionOnMapping

Controls how to check property definitions while mapping two extensible objects. See the "Object to Object Mapping" section to understand the 'CheckPairDefinitionOnMapping' option better.

Validation

You may want to add some **validation rules** for the extra properties you've defined. 'AddOrUpdateProperty' method options allows two ways of performing validation:

1. You can add **data annotation attributes** for a property.
2. You can write an action (code block) to perform a **custom validation**.

Validation works when you use the object in a method that is **automatically validated** (e.g. controller actions, page handler methods, application service methods...). So, all extra properties are validated whenever the extended object is being validated.

Data Annotation Attributes

All of the standard data annotation attributes are valid for extra properties. Example:

```
```csharp
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<IdentityUserCreateDto, string>(
 "SocialSecurityNumber",
 options =>
 {
 options.Attributes.Add(new RequiredAttribute());
 options.Attributes.Add(
 new StringLengthAttribute(32) {
 MinimumLength = 6
 }
);
 });
...;
```

With this configuration, `IdentityUserCreateDto` objects will be invalid without a valid `SocialSecurityNumber` value provided.

### #### Default Validation Attributes

There are some attributes \*\*automatically added\*\* when you create certain type of properties;

- \* `RequiredAttribute` is added for non nullable primitive property types (e.g. `int`, `bool`, `DateTime`...) and `enum` types.
- \* `EnumDataTypeAttribute` is added for enum types, to prevent to set invalid enum values.

Use `options.Attributes.Clear();` if you don't want these attributes.

### ### Custom Validation

If you need, you can add a custom action that is executed to validate the extra properties. Example:

```
```csharp
ObjectExtensionManager.Instance
    .AddOrUpdateProperty<IdentityUserCreateDto, string>(
        "SocialSecurityNumber",
        options =>
    {
        options.Validators.Add(context =>
    {
        var socialSecurityNumber = context.Value as string;

        if (socialSecurityNumber == null ||
            socialSecurityNumber.StartsWith("X"))
        {
            context.ValidationErrors.Add(
                new ValidationResult(
                    "Invalid social security number: " +
socialSecurityNumber,
                context));
        }
    });
});
```

```

                new[] { "SocialSecurityNumber" }
            );
        );
    });
...
);
```
`context.ServiceProvider` can be used to resolve a service dependency for advanced scenarios.

```

In addition to add custom validation logic for a single property, you can add a custom validation logic that is executed in object level. Example:

```

```csharp
ObjectExtensionManager.Instance
.AddOrUpdate<IdentityUserCreateDto>(objConfig =>
{
    //Define two properties with their own validation rules

    objConfig.AddOrUpdateProperty<string>("Password", propertyConfig =>
    {
        propertyConfig.Attributes.Add(new RequiredAttribute());
    });

    objConfig.AddOrUpdateProperty<string>("PasswordRepeat", propertyConfig =>
    {
        propertyConfig.Attributes.Add(new RequiredAttribute());
    });

    //Write a common validation logic works on multiple properties

    objConfigValidators.Add(context =>
    {
        if (context.ValidatingObject.GetProperty<string>("Password") !=
            context.ValidatingObject.GetProperty<string>("PasswordRepeat"))
        {
            context.ValidationErrors.Add(
                new ValidationResult(
                    "Please repeat the same password!",
                    new[] { "Password", "PasswordRepeat" }
                )
            );
        }
    });
});
```
Object to Object Mapping

```

Assume that you've added an extra property to an extensible entity object and used auto [object to object mapping](Object-To-Object-Mapping.md) to map this entity to an extensible DTO class. You need to be careful in such a case, because the extra property may contain a \*\*sensitive data\*\* that should not be available to clients.

This section offers some \*\*good practices\*\* to control your extra properties on object mapping.

### ### MapExtraPropertiesTo

`MapExtraPropertiesTo` is an extension method provided by the ABP Framework to copy extra properties from an object to another in a controlled manner. Example usage:

```
```csharp
identityUser.MapExtraPropertiesTo(identityUserDto);
````
```

`MapExtraPropertiesTo` \*\*requires to define properties\*\* (as described above) in \*\*both sides\*\* (`IdentityUser` and `IdentityUserDto` in this case) in order to copy the value to the target object. Otherwise, it doesn't copy the value even if it does exists in the source object (`identityUser` in this example). There are some ways to overload this restriction.

### #### MappingPropertyDefinitionChecks

`MapExtraPropertiesTo` gets an additional parameter to control the definition check for a single mapping operation:

```
```csharp
identityUser.MapExtraPropertiesTo(
    identityUserDto,
    MappingPropertyDefinitionChecks.None
);
````
```

> Be careful since `MappingPropertyDefinitionChecks.None` copies all extra properties without any check. `MappingPropertyDefinitionChecks` enum has other members too.

If you want to completely disable definition check for a property, you can do it while defining the extra property (or update an existing definition) as shown below:

```
```csharp
ObjectExtensionManager.Instance
    .AddOrUpdateProperty<IdentityUser, string>(
        "SocialSecurityNumber",
        options =>
    {
        options.CheckPairDefinitionOnMapping = false;
    });
````
```

### #### Ignored Properties

You may want to ignore some properties on a specific mapping operation:

```
```csharp
identityUser.MapExtraPropertiesTo(
    identityUserDto,
    ignoredProperties: new[] {"MySensitiveProp"}
);
````
```

Ignored properties are not copied to the target object.

#### #### AutoMapper Integration

If you're using the [AutoMapper](<https://automapper.org/>) library, the ABP Framework also provides an extension method to utilize the `MapExtraPropertiesTo` method defined above.

You can use the `MapExtraProperties()` method inside your mapping profile.

```
```csharp
public class MyProfile : Profile
{
    public MyProfile()
    {
        CreateMap<IdentityUser, IdentityUserDto>()
            .MapExtraProperties();
    }
}
...```

```

It has the same parameters with the `MapExtraPropertiesTo` method.

Entity Framework Core Database Mapping

If you're using the EF Core, you can map an extra property to a table field in the database. Example:

```
```csharp
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<IdentityUser, string>(
 "SocialSecurityNumber",
 options =>
 {
 options.MapEfCore(b => b.HasMaxLength(32));
 }
);
...```

```

See the [Entity Framework Core Integration document](Entity-Framework-Core.md) for more.

#### ## See Also

- \* [Module Entity Extensions](Module-Entity-Extensions.md)

## 6.11 Options

### # Options

Microsoft has introduced [the options pattern](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options>) that is used to configure a group of settings used by the framework services. This pattern is implemented by the [Microsoft.Extensions.Options](<https://www.nuget.org/packages/Microsoft.Extensions.Options>) package.

sions.Options) NuGet package, so it is usable by any type of applications in addition to ASP.NET Core based applications.

ABP framework follows this option pattern and defines options classes to configure the framework and the modules (they are explained in the documents of the related feature).

Since [the Microsoft documentation](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options>) explains the pattern in detail, no reason to repeat all. However, ABP adds a few more features and they will be explained here.

## ## Configure Options

You typically configure options in the `ConfigureServices` of the `Startup` class. However, since ABP framework provides a modular infrastructure, you configure options in the `ConfigureServices` of your [module](Module-Development-Basics.md). Example:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    context.Services.Configure<AbpAuditingOptions>(options =>
    {
        options.IsEnabled = false;
    });
}...
```

* `AbpAuditingOptions` is a simple class defines some properties like `Enabled` used here.
* `AbpModule` base class defines `Configure` method to make the code simpler. So, instead of `context.Services.Configure<...>`, you can directly use the `Configure<...>` shortcut method.

If you are developing a reusable module, you may need to define an options class to allow developers to configure your module. In this case, define a plain options class as shown below:

```
```csharp
public class MyOptions
{
 public int Value1 { get; set; }
 public bool Value2 { get; set; }
}...
```

Then developers can configure your options just like the `AbpAuditingOptions` example above:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    Configure<MyOptions>(options =>
    {
        options.Value1 = 42;
        options.Value2 = true;
});
```

```

} ...

* In this example, used the shortcut `Configure<...>` method.

### Get the Option Value

Whenever you need to get the value of an option, [inject](Dependency-Injection.md) the `IOptions<TOption>` service into your class and use its `.Value` property. Example:

```csharp
public class MyService : ITransientDependency
{
 private readonly MyOptions _options;

 public MyService(IOptions<MyOptions> options)
 {
 _options = options.Value; //Notice the options.Value usage!
 }

 public void DoIt()
 {
 var v1 = _options.Value1;
 var v2 = _options.Value2;
 }
}
```

```

Read [[the Microsoft documentation](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options)](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/options) for all details of the options pattern.

Pre Configure

One restriction of the options pattern is that you can only resolve (inject) the `IOptions<MyOptions>` and get the option values when the dependency injection configuration completes (that means the `ConfigureServices` methods of all modules complete).

If you are developing a module, you may need to allow developers to set some options and use these options in the dependency injection registration phase. You may need to configure other services or change the dependency injection registration code based on these option values.

For such cases, ABP introduces the `PreConfigure<TOptions>` and the `ExecutePreConfiguredActions<TOptions>` extension methods for the `IServiceCollection`. The pattern works as explained below.

Define a pre option class in your module. Example:

```

```csharp
public class MyPreOptions
{
 public bool MyValue { get; set; }
}
```

```

Then any [module class](Module-Development-Basics.md) depends on your module can use the `PreConfigure<TOptions>` method in its `PreConfigureServices` method. Example:

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
 PreConfigure<MyPreOptions>(options =>
 {
 options.MyValue = true;
 });
}...
```

> Multiple modules can pre-configure the options and override the option values based on their dependency order.

Finally, your module can execute the `ExecutePreConfiguredActions` method in its `ConfigureServices` method to get the configured option values. Example:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    var options =
context.Services.ExecutePreConfiguredActions<MyPreOptions>();
    if (options.MyValue)
    {
        //...
    }
}...
```

6.12 Settings

Settings

[Configuration system](Configuration.md) is a good way to configure the application on startup. In addition to the configurations, ABP provides another way to set and get some application settings.

A setting is a name-value pair stored in a dynamic data source, generally in a database. Setting system is extensible and there are pre-built providers for a user, a tenant, global and default.

Defining Settings

A setting must be defined before its use. ABP was designed to be [modular](Module-Development-Basics.md), so different modules can have different settings. A module must create a class derived from the `SettingDefinitionProvider` in order to define its settings. An example setting definition provider is shown below:

```
```csharp
public class EmailSettingProvider : SettingDefinitionProvider
```

```

{
 public override void Define(ISettingDefinitionContext context)
 {
 context.Add(
 new SettingDefinition("Smtp.Host", "127.0.0.1"),
 new SettingDefinition("Smtp.Port", "25"),
 new SettingDefinition("Smtp.UserName"),
 new SettingDefinition("Smtp.Password", isEncrypted: true),
 new SettingDefinition("Smtp.EnableSsl", "false")
);
 }
}
...

```

ABP automatically discovers this class and registers the setting definitions.

### **### SettingDefinition**

'SettingDefinition' class has the following properties:

- \* **\*\*Name\*\***: Unique name of the setting in the application. This is **the only mandatory property**. Used to get/set the value of this setting in the application code (It's a good idea to define a const string for a setting name instead of using a magic string).
- \* **\*\*DefaultValue\*\***: A setting may have a default value.
- \* **\*\*DisplayName\*\***: A localizable string that can be used to show the setting name on the UI.
- \* **\*\*Description\*\***: A localizable string that can be used to show the setting description on the UI.
- \* **\*\*IsVisibleToClients\*\***: A boolean value indicates that whether this setting value is available in the client side or not. Default value is false to prevent accidentally publishing an internal critical setting value.
- \* **\*\*IsInherited\*\***: A boolean value indicates that whether this setting value is inherited from other providers or not. Default value is true and fallbacks to the next provider if the setting value was not set for the requested provider (see the setting value providers section for more).
- \* **\*\*IsEncrypted\*\***: A boolean value indicates that whether this setting value should be encrypted on save and decrypted on read. It makes possible to secure the setting value in the database.
- \* **\*\*Providers\*\***: Can be used to restrict providers available for a particular setting (see the setting value providers section for more).
- \* **\*\*Properties\*\***: A name/value collection to set custom properties about this setting those can be used later in the application code.

### **### Change Setting Definitions of a Depended Module**

In some cases, you may want to change some properties of a settings defined in some other module that your application/module depends on. A setting definition provider can query and update setting definitions.

The following example gets a setting defined by the [\[Volo.Abp.Emailing\]](#)(Emailing.md) package and changes its properties:

```
```csharp
public class MySettingDefinitionProvider : SettingDefinitionProvider
{
    public override void Define(ISettingDefinitionContext context)
    {
```

```

        var smtpHost = context.GetOrNull("Abp.Mailing.Smtp.Host");
        if (smtpHost != null)
        {
            smtpHost.DefaultValue = "mail.mydomain.com";
            smtpHost.DisplayName =
                new LocalizableString(
                    typeof(MyLocalizationResource),
                    "SmtpServer_DisplayName"
                );
        }
    }
}

> Using constants for the setting names is a good practice and ABP packages do it. `Abp.Mailing.Smtp.Host` setting name is a constant defined by the `EmailSettingNames` class (in the `Volo.Abp.Emailing` namespace).

## Reading the Setting Values

### ISettingProvider

`ISettingProvider` is used to get the value of a setting or get the values of all the settings. Example usages:

```csharp
public class MyService
{
 private readonly ISettingProvider _settingProvider;

 //Inject ISettingProvider in the constructor
 public MyService(ISettingProvider settingProvider)
 {
 _settingProvider = settingProvider;
 }

 public async Task FooAsync()
 {
 //Get a value as string.
 string userName = await
 _settingProvider.GetOrNullAsync("Smtp.UserName");

 //Get a bool value and fallback to the default value (false) if not set.
 bool enableSsl = await
 _settingProvider.GetAsync<bool>("Smtp.EnableSsl");

 //Get a bool value and fallback to the provided default value (true) if not set.
 bool enableSsl = await _settingProvider.GetAsync<bool>(
 "Smtp.EnableSsl", defaultValue: true);

 //Get a bool value with the IsTrueAsync shortcut extension method
 bool enableSsl = await
 _settingProvider.IsTrueAsync("Smtp.EnableSsl");

 //Get an int value or the default value (0) if not set
 int port = (await _settingProvider.GetAsync<int>("Smtp.Port"));
 }
}
```

```

```
        //Get an int value or null if not provided
        int? port = (await
_settingProvider.GetOrNullAsync("Smtp.Port"))?.To<int>();
    }
}
```

> `ISettingProvider` is a very common service and some base classes (like `IApplicationService`) already property-inject it. You can directly use the `SettingProvider` property in such cases.

Reading Setting Values on the Client Side

If a setting is allowed to be visible on the client side, current value of the setting can also be read from the client code. See the following documents to understand how to get the setting values in different UI types;

- * [MVC / Razor Pages](UI/AspNetCore/JavaScript-API/Settings.md)
- * [Angular](UI/Angular/Settings.md)
- * [Blazor](UI/Blazor/Settings.md)

Setting Value Providers

Setting system is extensible, you can extend it by defining setting value providers to get setting values from any source and based on any condition.

`ISettingProvider` uses the setting value providers to obtain a setting value. It fallbacks to the next value provider if a value provider can not get the setting value.

There are 5 pre-built setting value providers registered by the order below:

- * `DefaultValueSettingValueProvider`: Gets the value from the default value of the setting definition, if set (see the `SettingDefinition` section above).
- * `ConfigurationSettingValueProvider`: Gets the value from the `[IConfiguration service]`(Configuration.md).
- * `GlobalSettingValueProvider`: Gets the global (system-wide) value for a setting, if set.
- * `TenantSettingValueProvider`: Gets the setting value for the current tenant, if set (see the `[multi-tenancy]`(Multi-Tenancy.md) document).
- * `UserSettingValueProvider`: Gets the setting value for the current user, if set (see the `[current user]`(CurrentUser.md) document).

> Setting fallback system works from bottom (user) to top (default).

Global, Tenant and User setting value providers use the `ISettingStore` to read the value from the data source (see the section below).

Setting Values in the Application Configuration

As mentioned in the previous section, `ConfigurationSettingValueProvider` reads the settings from the `IConfiguration` service, which can read values from the `appsettings.json` by default. So, the easiest way to configure setting values to define them in the `appsettings.json` file.

For example, you can configure `[IEmailSender]`(Emailing.md) settings as shown below:

```

```json
{
 "Settings": {
 "Abp.Mailing.DefaultFromAddress": "noreply@mydomain.com",
 "Abp.Mailing.DefaultFromDisplayName": "My Application",
 "Abp.Mailing.Smtp.Host": "mail.mydomain.com",
 "Abp.Mailing.Smtp.Port": "547",
 "Abp.Mailing.Smtp.UserName": "myusername",
 "Abp.Mailing.Smtp.Password": "mySecretPassW00rd",
 "Abp.Mailing.Smtp.EnableSsl": "True"
 }
}
```

```

Setting values should be configured under the `Settings` section as like in this example.

> `IConfiguration` is an .NET Core service and it can read values not only from the `appsettings.json`, but also from the environment, user secrets... etc. See [Microsoft's documentation](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/>) for more.

Custom Setting Value Providers

If you need to extend the setting system, you can define a class derived from the `SettingValueProvider` class. Example:

```

```csharp
public class CustomSettingValueProvider : SettingValueProvider
{
 public override string Name => "Custom";

 public CustomSettingValueProvider(ISettingStore settingStore)
 : base(settingStore)
 {
 }

 public override Task<string> GetOrNullAsync(SettingDefinition setting)
 {
 /* Return the setting value or null
 Use the SettingStore or another data source */
 }
}
```

```

> Alternatively, you can implement the `ISettingValueProvider` interface. Remember to register it to the [dependency injection](Dependency-Injection.md) in this case.

Every provider should have a unique Name (which is "Custom" here). Built-in providers use the given names:

- * `DefaultValueSettingValueProvider` : "D".
- * `ConfigurationSettingValueProvider` : "C".
- * `GlobalSettingValueProvider` : "G".
- * `TenantSettingValueProvider` : "T".
- * `UserSettingValueProvider` : "U".

One-letter names were preferred to reduce the data size in the database (provider name is repeated in each row).

Once you define a custom setting value provider, you need to explicitly register it to the `AbpSettingOptions`:

```
```csharp
Configure<AbpSettingOptions>(options =>
{
 options.ValueProviders.Add<CustomSettingValueProvider>();
});
```

This example adds it as the last item, so it will be the first value provider used by the `ISettingProvider`. You could add it to another index in the `options.ValueProviders` list.

### ### ISettingStore

While a setting value provider is free to use any source to get the setting value, the `ISettingStore` service is the default source of the setting values. Global, Tenant and User setting value providers use it.

### ## ISettingEncryptionService

`ISettingEncryptionService` is used to encrypt/decrypt setting values when `IsEncrypted` property of a setting definition was set to `true`.

You can replace this service in the dependency injection system to customize the encryption/decryption process. Default implementation uses the `StringEncryptionService` which is implemented with the AES algorithm by default (see string [encryption document](String-Encryption.md) for more).

### ## Setting Management Module

The core setting system is pretty independent and doesn't make any assumption about how you manage (change) the setting values. Even the default `ISettingStore` implementation is the `NullSettingStore` which returns null for all setting values.

The setting management module completes it (and implements `ISettingStore`) by managing setting values in a database. See the [Setting Management Module document](Modules/Setting-Management.md) for more.

## 6.13 Validation

### # Validation

Validation system is used to validate the user input or client request for a particular controller action or service method.

ABP is compatible with the ASP.NET Core Model Validation system and everything written in [its documentation](https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation) is already valid for ABP based

applications. So, this document mostly focuses on the ABP features rather than repeating the Microsoft documentation.

In addition, ABP adds the following benefits:

- \* Defines '`IValidationEnabled`' to add automatic validation to an arbitrary class. Since all the [application services](Application-Services.md) inherently implements it, they are also validated automatically.
- \* Automatically localize the validation errors for the data annotation attributes.
- \* Provides extensible services to validate a method call or an object state.
- \* Provides [FluentValidation](<https://fluentvalidation.net/>) integration.

### ## Validating DTOs

This section briefly introduces the validation system. For details, see the [ASP.NET Core validation documentation](<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation>).

### ### Data Annotation Attributes

Using data annotations is a simple way to implement the formal validation for a [DTO](Data-Transfer-Objects.md) in a declarative way. Example:

```
```csharp
public class CreateBookDto
{
    [Required]
    [StringLength(100)]
    public string Name { get; set; }

    [Required]
    [StringLength(1000)]
    public string Description { get; set; }

    [Range(0, 999.99)]
    public decimal Price { get; set; }
}
...```

```

When you use this class as a parameter to an [application service](Application-Services.md) or a controller, it is automatically validated and a localized validation exception is thrown ([and handled](Exception-Handling.md) by the ABP framework).

IValidatableObject

'`IValidatableObject`' can be implemented by a DTO to perform custom validation logic. '`CreateBookDto`' in the following example implements this interface and checks if the '`Name`' is equals to the '`Description`' and returns a validation error in this case.

```
```csharp
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace Acme.BookStore
{```

```

```

public class CreateBookDto : IValidatableObject
{
 [Required]
 [StringLength(100)]
 public string Name { get; set; }

 [Required]
 [StringLength(1000)]
 public string Description { get; set; }

 [Range(0, 999.99)]
 public decimal Price { get; set; }

 public IEnumerable<ValidationResult> Validate(
 ValidationContext validationContext)
 {
 if (Name == Description)
 {
 yield return new ValidationResult(
 "Name and Description can not be the same!",
 new[] { "Name", "Description" });
 }
 }
}
...

```

#### #### Resolving a Service

If you need to resolve a service from the [dependency injection system](Dependency-Injection.md), you can use the `ValidationContext` object. Example:

```

```csharp
var myService = validationContext.GetRequiredService<IMyService>();

> While resolving services in the `Validate` method allows any possibility,
it is not a good practice to implement your domain validation logic in DTOs.
Keep DTOs simple. Their purpose is to transfer data (DTO: Data Transfer
Object).

```

Validation Infrastructure

This section explains a few additional services provided by the ABP framework.

IValidationEnabled Interface

`IValidationEnabled` is an empty marker interface that can be implemented by any class (registered to and resolved from the [DI](Dependency-Injection.md)) to let the ABP framework perform the validation system for the methods of the class. Example:

```

```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;

```

```

using Volo.Abp.Validation;

namespace Acme.BookStore
{
 public class MyService : ITransientDependency, IValidationEnabled
 {
 public virtual async Task DoItAsync(MyInput input)
 {
 //...
 }
 }
}
```

```

> ABP framework uses the [dynamic proxying / interception](Dynamic-Proxying-Interceptors.md) system to perform the validation. In order to make it working, your method should be ****virtual**** or your service should be injected and used over an ****interface**** (like `'IMyService'`).

Enabling/Disabling Validation

You can use the `'[DisableValidation]'` to disable it for methods, classes and properties.

```

```csharp
[DisableValidation]
public Void MyMethod()
{
}

[DisableValidation]
public class InputClass
{
 public string MyProperty { get; set; }
}

public class InputClass
{
 [DisableValidation]
 public string MyProperty { get; set; }
}
```

```

AbpValidationException

Once ABP determines a validation error, it throws an exception of type `'AbpValidationException'`. Your application code can throw `'AbpValidationException'`, but most of the times it is not needed.

- * `'ValidationErrors'` property of the `'AbpValidationException'` contains the validation error list.
- * Log level of the `'AbpValidationException'` is set to `'Warning'`. It logs all the validation errors to the [logging system](Logging.md).
- * `'AbpValidationException'` is automatically caught by the ABP framework and converted to a usable error into with HTTP 400 status code. See the [exception handling](Exception-Handling.md) document for more.

Advanced Topics

IObjectValidator

In addition to the automatic validation, you may want to manually validate an object. In this case, [inject](Dependency-Injection.md) and use the `IObjectValidator` service:

- * `ValidateAsync` method validates the given object based on the validation rules and throws an `AbpValidationException` if it is not in a valid state.
- * `GetErrorsAsync` doesn't throw an exception, but only returns the validation errors.

`IObjectValidator` is implemented by the `ObjectValidator` by default. `ObjectValidator` is extensible; you can implement `IObjectValidationContributor` interface to contribute a custom logic.

Example:

```
```csharp
public class MyObjectValidationContributor
 : IObjectValidationContributor, ITransientDependency
{
 public Task AddErrorsAsync(ObjectValidationContext context)
 {
 //Get the validating object
 var obj = context.ValidatingObject;

 //Add the validation errors if available
 context.Errors.Add(...);
 return Task.CompletedTask;
 }
}
...```

```

- \* Remember to register your class to the [DI](Dependency-Injection.md) (implementing `ITransientDependency` does it just like in this example)
- \* ABP will automatically discover your class and use on any type of object validation (including automatic method call validation).

### ### IMethodInvocationValidator

`IMethodInvocationValidator` is used to validate a method call. It internally uses the `IObjectValidator` to validate objects passes to the method call. You normally don't need to this service since it is automatically used by the framework, but you may want to reuse or replace it on your application in rare cases.

### ## FluentValidation Integration

Volo.Abp.FluentValidation package integrates the FluentValidation library to the validation system (by implementing the `IObjectValidationContributor`). See the [FluentValidation Integration document](FluentValidation.md) for more.

#### 6.13.1 FluentValidation Integration

### # FluentValidation Integration

ABP [Validation](Validation.md) infrastructure is extensible.  
[Volo.Abp.FluentValidation](<https://www.nuget.org/packages/Volo.Abp.FluentValidation>) NuGet package extends the validation system to work with the [FluentValidation](<https://fluentvalidation.net/>) library.

## ## Installation

It is suggested to use the [ABP CLI](CLI.md) to install this package.

### ### Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.FluentValidation
````
```

### ### Manual Installation

If you want to manually install;

1. Add the  
[Volo.Abp.FluentValidation](<https://www.nuget.org/packages/Volo.Abp.FluentValidation>) NuGet package to your project:

```
```
Install-Package Volo.Abp.FluentValidation
````
```

2. Add the `AbpFluentValidationModule` to the dependency list of your module:

```
```csharp
[DependsOn(
    //...other dependencies
    typeof(AbpFluentValidationModule) //Add the FluentValidation module
)]
public class YourModule : AbpModule
{
}
````
```

## ## Using the FluentValidation

Follow [the FluentValidation documentation](<https://fluentvalidation.net/>) to create validator classes. Example:

```
```csharp
public class CreateUpdateBookDtoValidator :
AbstractValidator<CreateUpdateBookDto>
{
    public CreateUpdateBookDtoValidator()
    {
        RuleFor(x => x.Name).Length(3, 10);
        RuleFor(x => x.Price).ExclusiveBetween(0.0f, 999.0f);
    }
}
```

```
} ...
```

ABP will automatically find this class and associate with the '[CreateUpdateBookDto](#)' on object validation.

See Also

- * [\[Validation System\]](#)(Validation.md)

7 Infrastructure

7.1 Audit Logging

Audit Logging

[\[Wikipedia\]](#)(https://en.wikipedia.org/wiki/Audit_trail): "**An audit trail (also called **audit log**) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event**".

ABP Framework provides an **extensible audit logging system** that automates the audit logging by **convention** and provides **configuration** points to control the level of the audit logs.

An **audit log object** (see the Audit Log Object section below) is typically created & saved per web request. It includes;

- * **Request & response details** (like URL, Http method, Browser info, HTTP status code... etc.).
- * **Performed actions** (controller actions and application service method calls with their parameters).
- * **Entity changes** occurred in the web request.
- * **Exception** information (if there was an error while executing the request).
- * **Request duration** (to measure the performance of the application).

> [\[Startup templates\]](#)(Startup-Templates/Index.md) are configured for the audit logging system which is suitable for most of the applications. Use this document for a detailed control over the audit log system.

Database Provider Support

- * Fully supported by the [\[Entity Framework Core\]](#)(Entity-Framework-Core.md) provider.
- * Entity change logging is not supported by the [\[MongoDB\]](#)(MongoDB.md) provider. Other features work as expected.

UseAuditing()

`'UseAuditing()'` middleware should be added to the ASP.NET Core request pipeline in order to create and save the audit logs. If you've created your applications using [\[the startup templates\]](#)(Startup-Templates/Index.md), it is already added.

```

## AbpAuditingOptions

`AbpAuditingOptions` is the main [options object](Options.md) to configure the audit log system. You can configure it in the `ConfigureServices` method of your [module](Module-Development-Basics.md):

```csharp
Configure<AbpAuditingOptions>(options =>
{
 options.IsEnabled = false; //Disables the auditing system
});
```

```

Here, a list of the options you can configure:

- * `Enabled` (default: `true`): A root switch to enable or disable the auditing system. Other options are not used if this value is `false`.
- * `HideErrors` (default: `true`): Audit log system hides and writes regular [logs](Logging.md) if any error occurs while saving the audit log objects. If saving the audit logs is critical for your system, set this to `false` to throw exception in case of hiding the errors.
- * `EnabledForAnonymousUsers` (default: `true`): If you want to write audit logs only for the authenticated users, set this to `false`. If you save audit logs for anonymous users, you will see `null` for `UserId` values for these users.
- * `AlwaysLogOnException` (default: `true`): If you set to true, it always saves the audit log on an exception/error case without checking other options (except `Enabled`, which completely disables the audit logging).
- * `EnabledForIntegrationService` (default: `false`): Audit Logging is disabled for [integration services](Integration-Services.md) by default. Set this property as `true` to enable it.
- * `EnabledForGetRequests` (default: `false`): HTTP GET requests should not make any change in the database normally and audit log system doesn't save audit log objects for GET request. Set this to `true` to enable it also for the GET requests.
- * `DisableLogActionInfo` (default: `false`): If you set to true, Will no longer log `AuditLogActionInfo`.
- * `ApplicationName`: If multiple applications are saving audit logs into a single database, set this property to your application name, so you can distinguish the logs of different applications. If you don't set, it will set from the `IAplicationInfoAccessor.ApplicationName` value, which is the entry assembly name by default.
- * `IgnoredTypes`: A list of `Type`s to be ignored for audit logging. If this is an entity type, changes for this type of entities will not be saved. This list is also used while serializing the action parameters.
- * `EntityHistorySelectors`: A list of selectors those are used to determine if an entity type is selected for saving the entity change. See the section below for details.
- * `Contributors`: A list of `AuditLogContributor` implementations. A contributor is a way of extending the audit log system. See the "Audit Log Contributors" section below.
- * `AlwaysLogSelectors`: A list of selectors to save the audit logs for the matched criteria.

Entity History Selectors

Saving all changes of all your entities would require a lot of database space. For this reason, **audit log system doesn't save any change for the entities unless you explicitly configure it**.

To save all changes of all entities, simply use the `AddAllEntities()` extension method.

```
```csharp
Configure<AbpAuditingOptions>(options =>
{
 options.EntityHistorySelectors.AddAllEntities();
});
```

`options.EntityHistorySelectors` actually a list of type predicate. You can write a lambda expression to define your filter.

The example selector below does the same of the `AddAllEntities()` extension method defined above:

```
```csharp
Configure<AbpAuditingOptions>(options =>
{
    options.EntityHistorySelectors.Add(
        new NamedTypeSelector(
            "MySelectorName",
            type =>
            {
                if (typeof(IEntity).IsAssignableFrom(type))
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        )
    );
});
```

The condition `typeof(IEntity).IsAssignableFrom(type)` will be `true` for any class implements the `IEntity` interface (this is technically all the entities in your application). You can conditionally check and return `true` or `false` based on your preference.

`options.EntityHistorySelectors` is a flexible and dynamic way of selecting the entities for audit logging. Another way is to use the `Audited` and `DisableAuditing` attributes per entity.

AbpAspNetCoreAuditingOptions

`AbpAspNetCoreAuditingOptions` is the [options object](Options.md) to configure audit logging in the ASP.NET Core layer. You can configure it in the `ConfigureServices` method of your [module](Module-Development-Basics.md):

```
```csharp
Configure<AbpAspNetCoreAuditingOptions>(options =>
{
 options.IgnoredUrls.Add("/products");
});
```

```

`IgnoredUrls` is the only option. It is a list of ignored URLs prefixes. In the preceding example, all URLs starting with `/products` will be ignored for audit logging.

Enabling/Disabling Audit Logging for Services

Enable/Disable for Controllers & Actions

All the controller actions are logged by default (see `IsEnabledForGetRequests` above for GET requests).

You can use the `[DisableAuditing]` to disable it for a specific controller type:

```
```csharp
[DisableAuditing]
public class HomeController : AbpController
{
 //...
}
```

```

Use `[DisableAuditing]` for any action to control it in the action level:

```
```csharp
public class HomeController : AbpController
{
 [DisableAuditing]
 public async Task<ActionResult> Home()
 {
 //...
 }

 public async Task<ActionResult> OtherActionLogged()
 {
 //...
 }
}
```

```

Enable/Disable for Application Services & Methods

[Application service](Application-Services.md) method calls also included into the audit log by default. You can use the `[DisableAuditing]` in service or method level.

Enable/Disable for Other Services

Action audit logging can be enabled for any type of class (registered to and resolved from the [dependency injection](Dependency-Injection.md)) while it is only enabled for the controllers and the application services by default.

Use `[Audited]` and `[DisableAuditing]` for any class or method that need to be audit logged. In addition, your class can (directly or inherently) implement the `IAuditingEnabled` interface to enable the audit logging for that class by default.

Enable/Disable for Entities & Properties

An entity is ignored on entity change audit logging in the following cases;

- * If you add an entity type to the `AbpAuditingOptions.IgnoredTypes` (as explained before), it is completely ignored in the audit logging system.
- * If the object is not an [entity](Entities.md) (not implements `IEntity` directly or inherently - All entities implement this interface by default).
- * If entity type is not public.

Otherwise, you can use `Audited` to enable entity change audit logging for an entity:

```
```csharp
[Audited]
public class MyEntity : Entity<Guid>
{
 //...
}
...```

```

Or disable it for an entity:

```
```csharp
[DisableAuditing]
public class MyEntity : Entity<Guid>
{
    //...
}
...```

```

Disabling audit logging can be necessary only if the entity is being selected by the `AbpAuditingOptions.EntityHistorySelectors` that explained before.

You can disable auditing only some properties of your entities for a detailed control over the audit logging:

```
```csharp
[Audited]
public class MyUser : Entity<Guid>
{
 public string Name { get; set; }

 public string Email { get; set; }

 [DisableAuditing] //Ignore the Password on audit logging
 public string Password { get; set; }
}
...```

```

Audit log system will save changes for the `MyUser` entity while it ignores the `Password` property which can be dangerous to save for security purposes.

In some cases, you may want to save a few properties but ignore all others. Writing `[DisableAuditing]` for all the other properties would be tedious. In such cases, use `[Audited]` only for the desired properties and mark the entity with the `[DisableAuditing]` attribute:

```
```csharp
[DisableAuditing]
public class MyUser : Entity<Guid>
{
    [Audited] //Only log the Name change
    public string Name { get; set; }

    public string Email { get; set; }

    public string Password { get; set; }
}...```

```

IAuditingStore

`IAuditingStore` is an interface that is used to save the audit log objects (explained below) by the ABP Framework. If you need to save the audit log objects to a custom data store, you can implement the `IAuditingStore` in your own application and replace using the [\[dependency injection system\]](#)(Dependency-Injection.md).

`SimpleLogAuditingStore` is used if no audit store was registered. It simply writes the audit object to the standard [\[logging system\]](#)(Logging.md).

[\[The Audit Logging Module\]](#)(Modules/Audit-Logging.md) has been configured in [\[the startup templates\]](#)(Startup-Templates/Index.md) saves audit log objects to a database (it supports multiple database providers). So, most of the times you don't care about how `IAuditingStore` was implemented and used.

Audit Log Object

An **audit log object** is created for each **web request** by default. An audit log object can be represented by the following relation diagram:

![[**auditlog-object-diagram**]](images/auditlog-object-diagram.png)

- * **AuditLogInfo**: The root object with the following properties:
 - * `ApplicationName`: When you save audit logs of different applications to the same database, this property is used to distinguish the logs of the applications.
 - * `UserId`: Id of the current user, if the user has logged in.
 - * `UserName`: User name of the current user, if the user has logged in (this value is here to not depend on the identity module/system for lookup).
 - * `TenantId`: Id of the current tenant, for a multi-tenant application.
 - * `TenantName`: Name of the current tenant, for a multi-tenant application.
 - * `ExecutionTime`: The time when this audit log object has been created.
 - * `ExecutionDuration`: Total execution duration of the request, in milliseconds. This can be used to observe the performance of the application.
 - * `ClientId`: Id of the current client, if the client has been authenticated. A client is generally a 3rd-party application using the system over an HTTP API.
 - * `ClientName`: Name of the current client, if available.

- * `'ClientIpAddress'`: IP address of the client/user device.
- * `'CorrelationId'`: Current [Correlation Id](CorrelationId.md). Correlation Id is used to relate the audit logs written by different applications (or microservices) in a single logical operation.
- * `'BrowserInfo'`: Browser name/version info of the current user, if available.
- * `'HttpMethod'`: HTTP method of the current request (GET, POST, PUT, DELETE... etc.).
- * `'HttpStatuscode'`: HTTP response status code for this request.
- * `'Url'`: URL of the request.

* **AuditLogActionInfo**:** An audit log action is typically a controller action or an [application service](Application-Services.md) method call during the web request. One audit log may contain multiple actions. An action object has the following properties:

- * `'ServiceName'`: Name of the executed controller/service.
- * `'MethodName'`: Name of the executed method of the controller/service.
- * `'Parameters'`: A JSON formatted text representing the parameters passed to the method.
- * `'ExecutionTime'`: The time when this method was executed.
- * `'ExecutionDuration'`: Duration of the method execution, in milliseconds. This can be used to observe the performance of the method.

* **EntityChangeInfo**:** Represents a change of an entity in this web request. An audit log may contain zero or more entity changes. An entity change has the following properties:

- * `'ChangeTime'`: The time when the entity was changed.
- * `'ChangeType'`: An enum with the following fields: `'Created'` (0), `'Updated'` (1) and `'Deleted'` (2).
- * `'EntityId'`: Id of the entity that was changed.
- * `'EntityTenantId'`: Id of the tenant this entity belongs to.
- * `'EntityTypeFullName'`: Type (class) name of the entity with full namespace (like `*Acme.BookStore.Book*` for the Book entity).

* **EntityPropertyChangeInfo**:** Represents a change of a property of an entity. An entity change info (explained above) may contain one or more property change with the following properties:

- * `'NewValue'`: New value of the property. It is `'null'` if the entity was deleted.
- * `'OriginalValue'`: Old/original value before the change. It is `'null'` if the entity was newly created.
- * `'PropertyName'`: The name of the property on the entity class.
- * `'.PropertyTypeFullName'`: Type (class) name of the property with full namespace.

* **Exception**:** An audit log object may contain zero or more exception. In this way, you can get a report of the failed requests.

* **Comment**:** An arbitrary string value to add custom messages to the audit log entry. An audit log object may contain zero or more comments.

In addition to the standard properties explained above, `'AuditLogInfo'`, `'AuditLogActionInfo'` and `'EntityChangeInfo'` objects implement the `'IHasExtraProperties'` interface, so you can add custom properties to these objects.

Audit Log Contributors

You can extend the auditing system by creating a class that is derived from the `'AuditLogContributor'` class which defines the `'PreContribute'` and the `'PostContribute'` methods.

The only pre-built contributor is the `AspNetCoreAuditLogContributor` class which sets the related properties for an HTTP request.

A contributor can set properties and collections of the `AuditLogInfo` class to add more information.

Example:

```
```csharp
public class MyAuditLogContributor : AuditLogContributor
{
 public override void PreContribute(AuditLogContributionContext context)
 {
 var currentUser =
context.ServiceProvider.GetRequiredService<ICurrentUser>();
 context.AuditInfo SetProperty(
 "MyCustomClaimValue",
 currentUser.FindClaimValue("MyCustomClaim")
);
 }

 public override void PostContribute(AuditLogContributionContext context)
 {
 context.AuditInfo.Comments.Add("Some comment...");
 }
}
```
* `context.ServiceProvider` can be used to resolve services from the [dependency injection](Dependency-Injection.md).
* `context.AuditInfo` can be used to access to the current audit log object to manipulate it.
```

After creating such a contributor, you must add it to the `AbpAuditingOptions CONTRIBUTORS` list:

```
```csharp
Configure<AbpAuditingOptions>(options =>
{
 options CONTRIBUTORS.Add(new MyAuditLogContributor());
});
```
## IAuditLogScope & IAuditingManager
```

This section explains the `IAuditLogScope` & `IAuditingManager` services for advanced use cases.

An **audit log scope** is an [ambient scope](Ambient-Context-Pattern.md) that **builds** and **saves** an audit log object (explained before). By default, an audit log scope is created for a web request by the Audit Log Middleware (see `UseAuditing()` section above).

Access to the Current Audit Log Scope

Audit log contributors, was explained above, is a global way of manipulating the audit log object. It is good if you can get a value from a service.

If you need to manipulate the audit log object in an arbitrary point of your application, you can access to the current audit log scope and get the current audit log object (independent of how the scope is managed). Example:

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IAuditingManager _auditingManager;

 public MyService(IAuditingManager auditingManager)
 {
 _auditingManager = auditingManager;
 }

 public async Task DoItAsync()
 {
 var currentAuditLogScope = _auditingManager.Current;
 if (currentAuditLogScope != null)
 {
 currentAuditLogScope.Log.Comments.Add(
 "Executed the MyService.DoItAsync method :)");
 }

 currentAuditLogScope.Log SetProperty("MyCustomProperty", 42);
 }
}
```
...
```

Always check if `_auditingManager.Current` is null or not, because it is controlled in an outer scope and you can't know if an audit log scope was created before calling your method.

Manually Create an Audit Log Scope

You rarely need to create a manual audit log scope, but if you need, you can create an audit log scope using the `IAuditingManager` as like in the following example:

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IAuditingManager _auditingManager;

 public MyService(IAuditingManager auditingManager)
 {
 _auditingManager = auditingManager;
 }

 public async Task DoItAsync()
 {
 using (var auditingScope = _auditingManager.BeginScope())
 {
 try
 {
 //Call other services...
 }
 catch (Exception ex)
 }
 }
}
```
...
```

```

        {
            //Add exceptions
            _auditingManager.Current.Log.Exceptions.Add(ex);
            throw;
        }
        finally
        {
            //Always save the log
            await auditingScope.SaveAsync();
        }
    }
}
```

```

You can call other services, they may call others, they may change entities and so on. All these interactions are saved as a single audit log object in the finally block.

## ## The Audit Logging Module

The Audit Logging Module basically implements the `IAuditingStore` to save the audit log objects to a database. It supports multiple database providers. This module is added to the startup templates by default.

See [[the Audit Logging Module document](#)](Modules/Audit-Logging.md) for more about it.

## 7.2 Background Jobs

### # Background Jobs

#### ## Introduction

Background jobs are used to queue some tasks to be executed in the background. You may need background jobs for several reasons. Here are some examples:

- To perform \*\*long-running tasks\*\* without having the users wait. For example, a user presses a 'report' button to start a long-running reporting job. You add this job to the \*\*queue\*\* and send the report's result to your user via email when it's completed.
- To create \*\*re-trying\*\* and \*\*persistent tasks\*\* to \*\*guarantee\*\* that a code will be \*\*successfully executed\*\*. For example, you can send emails in a background job to overcome \*\*temporary failures\*\* and \*\*guarantee\*\* that it eventually will be sent. That way users do not wait while sending emails.

Background jobs are \*\*persistent\*\* that means they will be \*\*re-tried\*\* and \*\*executed\*\* later even if your application crashes.

#### ## Abstraction Package

ABP provides an \*\*abstraction\*\* module and \*\*several implementations\*\* for background jobs. It has a built-in/default implementation as well as Hangfire, RabbitMQ and Quartz integrations.

`'Volo.Abp.BackgroundJobs.Abstractions'` NuGet package provides needed services to create background jobs and queue background job items. If your module only depend on this package, it can be independent from the actual implementation/integration.

> `'Volo.Abp.BackgroundJobs.Abstractions'` package is installed to the startup templates by default.

### ### Create a Background Job

A background job is a class that implements the `'IBackgroundJob<TArgs>'` interface or derives from the `'BackgroundJob<TArgs>'` class. `'TArgs'` is a simple plain C# class to store the job data.

This example is used to send emails in background. First, define a class to store arguments of the background job:

```
```csharp
namespace MyProject
{
    public class EmailSendingArgs
    {
        public string EmailAddress { get; set; }
        public string Subject { get; set; }
        public string Body { get; set; }
    }
}...```

```

Then create a background job class that uses an `'EmailSendingArgs'` object to send an email:

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.BackgroundJobs;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Emailing;

namespace MyProject
{
 public class EmailSendingJob
 : AsyncBackgroundJob<EmailSendingArgs>, ITransientDependency
 {
 private readonly IEmailSender _emailSender;

 public EmailSendingJob(IEmailSender emailSender)
 {
 _emailSender = emailSender;
 }

 public override async Task ExecuteAsync(EmailSendingArgs args)
 {
 await _emailSender.SendAsync(
 args.EmailAddress,
 args.Subject,
 args.Body
);
 }
}
```

```

```
    }
}
```

This job simply uses ``IEmailSender`` to send emails (see [[email sending document](#)](Emailing.md)).

> `'AsyncBackgroundJob'` is used to create a job needs to perform async calls. You can inherit from `'BackgroundJob<TJob>'` and override the `'Execute'` method if the method doesn't need to perform any async call.

Exception Handling

A background job should not hide exceptions. If it throws an exception, the background job is automatically re-tried after a calculated waiting time. Hide exceptions only if you don't want to re-run the background job for the current argument.

Cancelling Background Jobs

If your background task is cancellable, then you can use the standard [[Cancellation Token](#)](Cancellation-Token-Provider.md) system to obtain a `'CancellationToken'` to cancel your job when requested. See the following example that uses the `'ICancellationTokenProvider'` to obtain the cancellation token:

```
```csharp
using System;
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Threading;

namespace MyProject
{
 public class LongRunningJob : AsyncBackgroundJob<LongRunningJobArgs>, ITransientDependency
 {
 private readonly ICancellationTokenProvider
 _cancellationTokenProvider;

 public LongRunningJob(ICancellationTokenProvider
cancellationTokenProvider)
 {
 _cancellationTokenProvider = cancellationTokenProvider;
 }

 public override async Task ExecuteAsync(LongRunningJobArgs args)
 {
 foreach (var id in args.Ids)
 {

 _cancellationTokenProvider.Token.ThrowIfCancellationRequested();
 await ProcessAsync(id); // code omitted for brevity
 }
 }
 }
}
```

```
...
```

> A cancellation operation might be needed if the application is shutting down and we don't want to block the application in the background job. This example throws an exception if the cancellation is requested. So, the job will be retried the next time the application starts. If you don't want that, just return from the `ExecuteAsync` method without throwing any exception (you can simply check the `_cancellationTokenProvider.Token.IsCancellationRequested` property).

#### #### Job Name

Each background job has a name. Job names are used in several places. For example, RabbitMQ provider uses job names to determine the RabbitMQ Queue names.

Job name is determined by the \*\*job argument type\*\*. For the `EmailSendingArgs` example above, the job name is `'MyProject.EmailSendingArgs'` (full name, including the namespace). You can use the `BackgroundJobName` attribute to set a different job name.

#### \*\*Example\*\*

```
```csharp
using Volo.Abp.BackgroundJobs;

namespace MyProject
{
    [BackgroundJobName("emails")]
    public class EmailSendingArgs
    {
        public string EmailAddress { get; set; }
        public string Subject { get; set; }
        public string Body { get; set; }
    }
}..
```

Queue a Job Item

Now, you can queue an email sending job using the `IBackgroundJobManager` service:

```
```csharp
public class RegistrationService : ApplicationService
{
 private readonly IBackgroundJobManager _backgroundJobManager;

 public RegistrationService(IBackgroundJobManager backgroundJobManager)
 {
 _backgroundJobManager = backgroundJobManager;
 }

 public async Task RegisterAsync(string userName, string emailAddress,
string password)
 {
 //TODO: Create new user in the database...
```

```

 await _backgroundJobManager.EnqueueAsync(
 new EmailSendingArgs
 {
 EmailAddress = emailAddress,
 Subject = "You've successfully registered!",
 Body = "..."
 }
);
 }
}
```

```

Just injected `IBackgroundJobManager` service and used its `EnqueueAsync` method to add a new job to the queue.

Enqueue method gets some optional arguments to control the background job:

- * **priority** is used to control priority of the job item. It gets an `BackgroundJobPriority` enum which has `Low`, `BelowNormal`, `Normal` (default), `AboveNormal` and `Hight` fields.
- * **delay** is used to wait a while (`TimeSpan`) before first try.

Disable Job Execution

You may want to disable background job execution for your application. This is generally needed if you want to execute background jobs in another process and disable it for the current process.

Use `AbpBackgroundJobOptions` to configure the job execution:

```

```csharp
[DependsOn(typeof(AbpBackgroundJobsModule))]
public class MyModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 Configure<AbpBackgroundJobOptions>(options =>
 {
 options.IsJobExecutionEnabled = false; //Disables job execution
 });
 }
}
```

```

Default Background Job Manager

ABP framework includes a simple `IBackgroundJobManager` implementation that;

- Works as **FIFO** in a **single thread**.
- **Retries** job execution until the job **successfully runs** or **timeouts**. Default timeout is 2 days for a job. Logs all exceptions.
- **Deletes** a job from the store (database) when it's successfully executed. If it's timed out, it sets it as **abandoned** and leaves it in the database.
- **Increasingly waits between retries** for a job. It waits 1 minute for the first retry, 2 minutes for the second retry, 4 minutes for the third retry and so on.

- ****Polls**** the store for jobs in fixed intervals. It queries jobs, ordering by priority (asc) and then by try count (asc).
- > `Volo.Abp.BackgroundJobs` nuget package contains the default background job manager and it is installed to the startup templates by default.

Configuration

Use `AbpBackgroundJobWorkerOptions` in your [module class](Module-Development-Basics.md) to configure the default background job manager. The example below changes the timeout duration for background jobs:

```
```csharp
[DependsOn(typeof(AbpBackgroundJobsModule))]
public class MyModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 Configure<AbpBackgroundJobWorkerOptions>(options =>
 {
 options.DefaultTimeout = 864000; //10 days (as seconds)
 });
 }
}
```

```

Data Store

The default background job manager needs a data store to save and read jobs. It defines `IBackgroundJobStore` as an abstraction to store the jobs.

Background Jobs module implements `IBackgroundJobStore` using various data access providers. See its own [documentation](Modules/Background-Jobs.md). If you don't want to use this module, you should implement the `IBackgroundJobStore` interface yourself.

> Background Jobs module is already installed to the startup templates by default and it works based on your ORM/data access choice.

Clustered Deployment

The default background job manager is compatible with [clustered environments](Deployment/Clustered-Environment.md) (where multiple instances of your application run concurrently). It uses a [distributed lock](Distributed-Locking.md) to ensure that the jobs are executed only in a single application instance at a time.

However, the distributed lock system works in-process by default. That means it is not distributed actually, unless you configure a distributed lock provider. So, **please follow the [distributed lock](Distributed-Locking.md) document to configure a provider for your application**, if it is not already configured.

If you don't want to use a distributed lock provider, you may go with the following options:

```
* Stop the background job manager (set  
`AbpBackgroundJobOptions.IsJobExecutionEnabled` to `false` as explained in  
the *Disable Job Execution* section) in all application instances except one  
of them, so only the single instance executes the jobs (while other  
application instances can still queue jobs).  
* Stop the background job manager (set  
`AbpBackgroundJobOptions.IsJobExecutionEnabled` to `false` as explained in  
the *Disable Job Execution* section) in all application instances and create  
a dedicated application (maybe a console application running in its own  
container or a Windows Service running in the background) to execute all the  
background jobs. This can be a good option if your background jobs consume  
high system resources (CPU, RAM or Disk), so you can deploy that background  
application to a dedicated server and your background jobs don't affect your  
application's performance.
```

Integrations

Background job system is extensible and you can change the default background
job manager with your own implementation or one of the pre-built integrations.

See pre-built job manager alternatives:

- * [\[Hangfire Background Job Manager\]](#)(Background-Jobs-Hangfire.md)
- * [\[RabbitMQ Background Job Manager\]](#)(Background-Jobs-RabbitMq.md)
- * [\[Quartz Background Job Manager\]](#)(Background-Jobs-Quartz.md)

See Also

- * [\[Background Workers\]](#)(Background-Workers.md)

7.2.1 Hangfire Integration

Hangfire Background Job Manager

[\[Hangfire\]](#)(<https://www.hangfire.io/>) is an advanced background job manager.
You can integrate Hangfire with the ABP Framework to use it instead of the
[\[default background job manager\]](#)(Background-Jobs.md). In this way, you can
use the same background job API for Hangfire and your code will be
independent of Hangfire. If you like, you can directly use Hangfire's API,
too.

> See the [\[background jobs document\]](#)(Background-Jobs.md) to learn how to use
the background job system. This document only shows how to install and
configure the Hangfire integration.

Installation

It is suggested to use the [\[ABP CLI\]](#)(CLI.md) to install this package.

Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and
type the following command:

```
```bash  
abp add-package Volo.Abp.BackgroundJobs.HangFire
```
```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.BackgroundJobs.HangFire).

Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.BackgroundJobs.HangFire](https://www.nuget.org/packages/Volo.Abp.BackgroundJobs.HangFire) NuGet package to your project:

```
```
Install-Package Volo.Abp.BackgroundJobs.HangFire
```
```

2. Add the `AbpBackgroundJobsHangfireModule` to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundJobsHangfireModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
```
```

Configuration

You can install any storage for Hangfire. The most common one is SQL Server (see the [Hangfire.SqlServer](https://www.nuget.org/packages/Hangfire.SqlServer) NuGet package).

After you have installed these NuGet packages, you need to configure your project to use Hangfire.

1. First, we change the `Module` class (example: `<YourProjectName>HttpApiHostModule`) to add Hangfire configuration of the storage and connection string in the `ConfigureServices` method:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 var configuration = context.Services.GetConfiguration();
 var hostingEnvironment = context.Services.GetHostingEnvironment();

 //... other configurations.

 ConfigureHangfire(context, configuration);
}

private void ConfigureHangfire(ServiceConfigurationContext context,
IConfiguration configuration)
{
 context.Services.AddHangfire(config =>

```

```
 {
 config.UseSqlServerStorage(configuration.GetConnectionString("Default"));
 };
 };
};

> You have to configure a storage for Hangfire.

2. If you want to use hangfire's dashboard, you can add
'UseAbpHangfireDashboard' call in the 'OnApplicationInitialization' method in
'Module' class:
```

```
```csharp
public override void
OnApplicationInitialization(ApplicationInitializationContext context)
{
    var app = context.GetApplicationBuilder();
    // ... others

    app.UseAbpHangfireDashboard(); //should add to the request pipeline
before the app.UseConfiguredEndpoints()
    app.UseConfiguredEndpoints();
}
````
```

### ### Specifying Queue

You can use the `[`QueueAttribute`](https://docs.hangfire.io/en/latest/background-processing/configuring-queues.html)` to specify the queue:

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.BackgroundJobs;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Emailing;

namespace MyProject
{
    [Queue("alpha")]
    public class EmailSendingJob
        : AsyncBackgroundJob<EmailSendingArgs>, ITransientDependency
    {
        private readonly IEmailSender _emailSender;

        public EmailSendingJob(IEmailSender emailSender)
        {
            _emailSender = emailSender;
        }

        public override async Task ExecuteAsync(EmailSendingArgs args)
        {
            await _emailSender.SendAsync(
                args.EmailAddress,
                args.Subject,
                args.Body
            );
        }
    }
}
```

```
        );
    }
}...
```

Dashboard Authorization

Hangfire Dashboard provides information about your background jobs, including method names and serialized arguments as well as gives you an opportunity to manage them by performing different actions – retry, delete, trigger, etc. So it is important to restrict access to the Dashboard.

To make it secure by default, only local requests are allowed, however you can change this by following the [[official documentation](#)](<http://docs.hangfire.io/en/latest/configuration/using-dashboard.html>) of Hangfire.

You can integrate the Hangfire dashboard to [[ABP authorization system](#)](Authorization.md) using the `**AbpHangfireAuthorizationFilter**` class. This class is defined in the `'Volo.Abp.Hangfire'` package. The following example, checks if the current user is logged in to the application:

```
```csharp
app.UseAbpHangfireDashboard("/hangfire", options =>
{
 options.AsyncAuthorization = new[] { new
AbpHangfireAuthorizationFilter() };
});
```

\* `'AbpHangfireAuthorizationFilter'` is an implementation of an authorization filter.

#### #### AbpHangfireAuthorizationFilter

`'AbpHangfireAuthorizationFilter'` class has the following fields:

- \* `**'enableTenant'` (`bool`, default: `false`):**` Enables/disables accessing the Hangfire dashboard on tenant users.
- \* `**'requiredPermissionName'` (`string`, default: `null`):**` Hangfire dashboard is accessible only if the current user has the specified permission. In this case, if we specify a permission name, we don't need to set `'enableTenant'` 'true'` because the permission system already does it.

If you want to require an additional permission, you can pass it into the constructor as below:

```
```csharp
app.UseAbpHangfireDashboard("/hangfire", options =>
{
    options.AsyncAuthorization = new[] { new
AbpHangfireAuthorizationFilter(requiredPermissionName:
"MyHangFireDashboardPermissionName") };
});
```

****Important**:** `UseAbpHangfireDashboard` should be called after the authentication and authorization middlewares in your `Startup` class (probably at the last line). Otherwise, authorization will always fail!

7.2.2 RabbitMQ Integration

RabbitMQ Background Job Manager

RabbitMQ is an industry standard message broker. While it is typically used for inter-process communication (messaging / distributed events), it is pretty useful to store and execute background jobs in FIFO (First In First Out) order.

ABP Framework provides the [Volo.Abp.BackgroundJobs.RabbitMQ](<https://www.nuget.org/packages/Volo.Abp.BackgroundJobs.RabbitMQ>) NuGet package to use the RabbitMQ for background job execution.

> See the [background jobs document](Background-Jobs.md) to learn how to use the background job system. This document only shows how to install and configure the RabbitMQ integration.

Installation

Use the ABP CLI to add [Volo.Abp.BackgroundJobs.RabbitMQ](<https://www.nuget.org/packages/Volo.Abp.BackgroundJobs.RabbitMQ>) NuGet package to your project:

- * Install the [ABP CLI](<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- * Open a command line (terminal) in the directory of the `*.csproj` file you want to add the `Volo.Abp.BackgroundJobs.RabbitMQ` package.
- * Run `abp add-package Volo.Abp.BackgroundJobs.RabbitMQ` command.

If you want to do it manually, install the [Volo.Abp.BackgroundJobs.RabbitMQ](<https://www.nuget.org/packages/Volo.Abp.BackgroundJobs.RabbitMQ>) NuGet package to your project and add `[DependsOn(typeof(AbpBackgroundJobsRabbitMqModule))` to the [ABP module](Module-Development-Basics.md) class inside your project.

Configuration

Default Configuration

The default configuration automatically connects to the local RabbitMQ server (localhost) with the standard port. **In this case, no configuration needed.**

RabbitMQ Connection(s)

You can configure the RabbitMQ connections using the standard [configuration system](Configuration.md), like using the `appsettings.json` file, or using the [options](Options.md) classes.

`appsettings.json` file configuration

This is the simplest way to configure the RabbitMQ connections. It is also very strong since you can use any other configuration source (like environment variables) that is [supported by the AspNet Core](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/>).

****Example: Configuring the Default RabbitMQ Connection****

```
```json
{
 "RabbitMQ": {
 "Connections": {
 "Default": {
 "HostName": "123.123.123.123",
 "Port": "5672"
 }
 }
 }
}...```

```

You can use any of the [ConnectionFactory](<http://rabbitmq.github.io/rabbitmq-dotnet-client/api/RabbitMQ.Client.ConnectionFactory.html#properties>) properties as the connection properties. See [[the RabbitMQ document](https://www.rabbitmq.com/dotnet-api-guide.html#exchanges-and-queues)](<https://www.rabbitmq.com/dotnet-api-guide.html#exchanges-and-queues>) to understand these options better.

Defining multiple connections is allowed. In this case, you can use different connections for different background job types (see the 'AbpRabbitMqBackgroundJobOptions' section below).

**\*\*Example: Declare two connections\*\***

```
```json
{
  "RabbitMQ": {
    "Connections": {
      "Default": {
        "HostName": "123.123.123.123"
      },
      "SecondConnection": {
        "HostName": "321.321.321.321"
      }
    }
  }
}...```

```

If you need to connect to the RabbitMQ cluster, you can use the `;` character to separate the host names.

****Example: Connect to the RabbitMQ cluster****

```
```json
{
 "RabbitMQ": {
 "Connections": {```

```

```

 "Default": {
 "HostName": "123.123.123.123;234.234.234.234"
 }
 },
 "EventBus": {
 "ClientName": "MyClientName",
 "ExchangeName": "MyExchangeName"
 }
}
```

```

AbpRabbitMqOptions

`'AbpRabbitMqOptions'` class can be used to configure the connection strings for the RabbitMQ. You can configure this options inside the `'ConfigureServices'` of your `[module](Module-Development-Basics.md)`.

****Example: Configure the connection****

```

```csharp
Configure<AbpRabbitMqOptions>(options =>
{
 options.Connections.Default.UserName = "user";
 options.Connections.Default.Password = "pass";
 options.Connections.Default.HostName = "123.123.123.123";
 options.Connections.Default.Port = 5672;
});
```

```

Using these options classes can be combined with the `'appsettings.json'` way. Configuring an option property in the code overrides the value in the configuration file.

AbpRabbitMqBackgroundJobOptions

Job Queue Names

By default, each job type uses a separate queue. Queue names are calculated by combining a standard prefix and the job name. Default prefix is `'AbpBackgroundJobs.'` So, if the job name is `'EmailSending'` then the queue name in the RabbitMQ becomes `'AbpBackgroundJobs.EmailSending'`

> Use `'BackgroundJobName'` attribute on the background ****job argument**** class to specify the job name. Otherwise, the job name will be the full name (with namespace) of the job class.

Job Connections

By default, all the job types use the `'Default'` RabbitMQ connection.

Customization

`'AbpRabbitMqBackgroundJobOptions'` can be used to customize the queue names and the connections used by the jobs.

****Example:****

```

```csharp
Configure<AbpRabbitMqBackgroundJobOptions>(options =>
{
 options.DefaultQueueNamePrefix = "my_app_jobs.";
 options.DefaultDelayedQueueNamePrefix = "my_app_jobs.delayed"
 options.PrefetchCount = 1;
 options.JobQueues[typeof(EmailSendingArgs)] =
 new JobQueueConfiguration(
 typeof(EmailSendingArgs),
 queueName: "my_app_jobs.emails",
 connectionName: "SecondConnection",
 delayedQueueName:"my_app_jobs.emails.delayed"
);
});
```
* This example sets the default queue name prefix to `my_app_jobs.` and default delayed queue name prefix to `my_app_jobs.delayed`. If different applications use the same RabbitMQ server, it would be important to use different prefixes for each application to not consume jobs of each other.
* Sets `PrefetchCount` for all queues.
* Also specifies a different connection string for the `EmailSendingArgs`.

`JobQueueConfiguration` class has some additional options in its constructor;

* `queueName`: The queue name that is used for this job. The prefix is not added, so you need to specify the full name of the queue.
* `DelayedQueueName`: The delayed queue name that is used for delayed execution of job. The prefix is not added, so you need to specify the full name of the queue.
* `connectionName`: The RabbitMQ connection name (see the connection configuration above). This is optional and the default value is `Default`.
* `durable` (optional, default: `true`).
* `exclusive` (optional, default: `false`).
* `autoDelete` (optional, default: `false`).
* `PrefetchCount` (optional, default: null)

See the RabbitMQ documentation if you want to understand the `durable`, `exclusive` and `autoDelete` options better, while most of the times the default configuration is what you want.

```

See Also

- * [Background Jobs](Background-Jobs.md)

7.2.3 Quartz Integration

Quartz Background Job Manager

[Quartz](<https://www.quartz-scheduler.net/>) is an advanced background job manager. You can integrate Quartz with the ABP Framework to use it instead of the [default background job manager](Background-Jobs.md). In this way, you can use the same background job API for Quartz and your code will be independent of Quartz. If you like, you can directly use Quartz's API, too.

> See the [background jobs document](Background-Jobs.md) to learn how to use the background job system. This document only shows how to install and configure the Quartz integration.

Installation

It is suggested to use the [ABP CLI](CLI.md) to install this package.

Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.BackgroundJobs.Quartz
````
```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.BackgroundJobs.Quartz).

Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.BackgroundJobs.Quartz](https://www.nuget.org/packages/Volo.Abp.BackgroundJobs.Quartz) NuGet package to your project:

```
```
Install-Package Volo.Abp.BackgroundJobs.Quartz
````
```

2. Add the `AbpBackgroundJobsQuartzModule` to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundJobsQuartzModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
````
```

Configuration

Quartz is a very configurable library, and the ABP framework provides `AbpQuartzOptions` for this. You can use the `PreConfigure` method in your module class to pre-configure this option. ABP will use it when initializing the Quartz module. For example:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundJobsQuartzModule) //Add the new module dependency
)]
````
```

```

public class YourModule : AbpModule
{
    public override void PreConfigureServices(ServiceConfigurationContext context)
    {
        var configuration = context.Services.GetConfiguration();

        PreConfigure<AbpQuartzOptions>(options =>
        {
            options.Properties = new NameValueCollection
            {
                ["quartz.jobStore.dataSource"] = "BackgroundJobsDemoApp",
                ["quartz.jobStore.type"] =
"Quartz.Impl.AdoJobStore.JobStoreTX, Quartz",
                ["quartz.jobStore.tablePrefix"] = "QRTZ_",
                ["quartz.serializer.type"] = "json",
                ["quartz.dataSource.BackgroundJobsDemoApp.connectionString"] =
configuration.GetConnectionString("Quartz"),
                ["quartz.dataSource.BackgroundJobsDemoApp.provider"] =
"SqlServer",
                ["quartz.jobStore.driverDelegateType"] =
"Quartz.Impl.AdoJobStore.SqlServerDelegate, Quartz",
            };
        });
    }
}
```

```

Starting from ABP 3.1 version, we have added '`Configurator`' to '`AbpQuartzOptions`' to configure Quartz. For example:

```

```csharp
[DependsOn(
    //...other dependencies
    typeof(AbpBackgroundJobsQuartzModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
    public override void PreConfigureServices(ServiceConfigurationContext context)
    {
        var configuration = context.Services.GetConfiguration();

        PreConfigure<AbpQuartzOptions>(options =>
        {
            options.Configurator = configure =>
            {
                configure.UsePersistentStore(storeOptions =>
                {
                    storeOptions.UseProperties = true;
                    storeOptions.UseJsonSerializer();

                    storeOptions.UseSqlServer(configuration.GetConnectionString("Quartz"));
                    storeOptions.UseClustering(c =>
                    {
                        c.CheckinMisfireThreshold = TimeSpan.FromSeconds(20);
                        c.CheckinInterval = TimeSpan.FromSeconds(10);
                    });
                });
            };
        });
    }
}
```

```

```
 });
 });
}
```

```

> You can choose the way you favorite to configure Quaratz.

Quartz stores job and scheduling information **in memory by default**. In the example, we use the pre-configuration of [options pattern](Options.md) to change it to the database. For more configuration of Quartz, please refer to the Quartz's [documentation](<https://www.quartz-scheduler.net/>).

Exception handling

Default exception handling strategy

When an exception occurs in the background job, ABP provide the **default handling strategy** retrying once every 3 seconds, up to 3 times. You can change the retry count and retry interval via `AbpBackgroundJobQuartzOptions` options:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundJobsQuartzModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext
context)
 {
 Configure<AbpBackgroundJobQuartzOptions>(options =>
 {
 options.RetryCount = 1;
 options.RetryIntervalMillisecond = 1000;
 });
 }
}
```

```

Customize exception handling strategy

You can customize the exception handling strategy via `AbpBackgroundJobQuartzOptions` options:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundJobsQuartzModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext
context)
 {
 Configure<AbpBackgroundJobQuartzOptions>(options =>
```

```

 {
 options.RetryStrategy = async (retryIndex, executionContext,
exception) =>
 {
 // customize exception handling
 };
 });
 }
}..

```

### 7.3 Background Workers

#### # Background Workers

##### ## Introduction

Background workers are simple independent threads in the application running in the background. Generally, they run periodically to perform some tasks.

Examples;

- \* A background worker can run periodically to \*\*delete old logs\*\*.
- \* A background worker can run periodically to \*\*determine inactive users\*\* and \*\*send emails\*\* to get users to return to your application.

##### ## Create a Background Worker

A background worker should directly or indirectly implement the `IBackgroundWorker` interface.

> A background worker is inherently [singleton](Dependency-Injection.md). So, only a single instance of your worker class is instantiated and run.

##### ### BackgroundWorkerBase

`BackgroundWorkerBase` is an easy way to create a background worker.

```
```csharp
public class MyWorker : BackgroundWorkerBase
{
    public override Task StartAsync(CancellationToken cancellationToken =
default)
    {
        //...
    }

    public override Task StopAsync(CancellationToken cancellationToken =
default)
    {
        //...
    }
}..
```

```

Start your worker in the '`StartAsync`' (which is called when the application begins) and stop in the '`StopAsync`' (which is called when the application shuts down).

> You can directly implement the '`IBackgroundWorker`', but '`BackgroundWorkerBase`' provides some useful properties like '`Logger`'.

### ### `AsyncPeriodicBackgroundWorkerBase`

Assume that we want to make a user passive, if the user has not logged in to the application in last 30 days. '`AsyncPeriodicBackgroundWorkerBase`' class simplifies to create periodic workers, so we will use it for the example below:

```
```csharp
public class PassiveUserCheckerWorker : AsyncPeriodicBackgroundWorkerBase
{
    public PassiveUserCheckerWorker(
        AbpAsyncTimer timer,
        IServiceProvider serviceScopeFactory
    ) : base(
        timer,
        serviceScopeFactory)
    {
        Timer.Period = 600000; //10 minutes
    }

    protected override Task DoWorkAsync(
        PeriodicBackgroundWorkerContext workerContext)
    {
        Logger.LogInformation("Starting: Setting status of inactive
users...");

        //Resolve dependencies
        var userRepository = workerContext
            .ServiceProvider
            .GetRequiredService<IUserRepository>();

        //Do the work
        await userRepository.UpdateInactiveUserStatusesAsync();

        Logger.LogInformation("Completed: Setting status of inactive
users...");
    }
}
```

```

\* '`AsyncPeriodicBackgroundWorkerBase`' uses the '`AbpTimer`' (a thread-safe timer) object to determine \*\*the period\*\*. We can set its '`Period`' property in the constructor.

\* It required to implement the '`DoWorkAsync`' method to \*\*execute\*\* the periodic work.

\* It is a good practice to \*\*resolve dependencies\*\* from the '`PeriodicBackgroundWorkerContext`' instead of constructor injection. Because '`AsyncPeriodicBackgroundWorkerBase`' uses a '`IServiceScope`' that is \*\*disposed\*\* when your work finishes.

\* '`AsyncPeriodicBackgroundWorkerBase`' \*\*catches and logs exceptions\*\* thrown by the '`DoWorkAsync`' method.

## ## Register Background Worker

After creating a background worker class, you should add it to the `IBackgroundWorkerManager`. The most common place is the `OnApplicationInitializationAsync` method of your module class:

```
```csharp
[DependsOn(typeof(AbpBackgroundWorkersModule))]
public class MyModule : AbpModule
{
    public override async Task OnApplicationInitializationAsync(
        ApplicationInitializationContext context)
    {
        await context.AddBackgroundWorkerAsync<PassiveUserCheckerWorker>();
    }
}
````
```

`context.AddBackgroundWorkerAsync(...)` is a shortcut extension method for the expression below:

```
```csharp
await context.ServiceProvider
    .GetRequiredService<IBackgroundWorkerManager>()
    .AddAsync(
        context
            .ServiceProvider
            .GetRequiredService<PassiveUserCheckerWorker>()
    );
````
```

So, it resolves the given background worker and adds to the `IBackgroundWorkerManager`.

While we generally add workers in `OnApplicationInitializationAsync`, there are no restrictions on that. You can inject `IBackgroundWorkerManager` anywhere and add workers at runtime. Background worker manager will stop and release all the registered workers when your application is being shut down.

## ## Options

`AbpBackgroundWorkerOptions` class is used to [set options](Options.md) for the background workers. Currently, there is only one option:

- \* `IsEnabled` (default: true): Used to \*\*enable/disable\*\* the background worker system for your application.

> See the [Options](Options.md) document to learn how to set options.

## ## Making Your Application Always Run

Background workers only work if your application is running. If you host the background job execution in your web application (this is the default behavior), you should ensure that your web application is configured to always be running. Otherwise, background jobs only work while your application is in use.

## ## Running On a Cluster

Be careful if you run multiple instances of your application simultaneously in a clustered environment. In that case, every application runs the same worker which may create conflicts if your workers are running on the same resources (processing the same data, for example).

If that's a problem for your workers, you have the following options:

- \* Implement your background workers so that they work in a clustered environment without any problem. Using the [distributed lock](Distributed-Locking.md) to ensure concurrency control is a way of doing that. A background worker in an application instance may handle a distributed lock, so the workers in other application instances will wait for the lock. In this way, only one worker does the actual work, while others wait in idle. If you implement this, your workers run safely without caring about how the application is deployed.
- \* Stop the background workers (set '`AbpBackgroundWorkerOptions.IsEnabled`' to '`false`') in all application instances except one of them, so only the single instance runs the workers.
- \* Stop the background workers (set '`AbpBackgroundWorkerOptions.IsEnabled`' to '`false`') in all application instances and create a dedicated application (maybe a console application running in its own container or a Windows Service running in the background) to execute all the background tasks. This can be a good option if your background workers consume high system resources (CPU, RAM or Disk), so you can deploy that background application to a dedicated server and your background tasks don't affect your application's performance.

## ## Integrations

Background worker system is extensible and you can change the default background worker manager with your own implementation or one of the pre-built integrations.

See pre-built worker manager alternatives:

- \* [Quartz Background Worker Manager](Background-Workers-Quartz.md)
- \* [Hangfire Background Worker Manager](Background-Workers-Hangfire.md)

## ## See Also

- \* [Background Jobs](Background-Jobs.md)

### 7.3.1 Quartz Integration

#### # Quartz Background Worker Manager

[Quartz](<https://www.quartz-scheduler.net/>) is an advanced background worker manager. You can integrate Quartz with the ABP Framework to use it instead of the [default background worker manager](Background-Workers.md). ABP simply integrates quartz.

#### ## Installation

It is suggested to use the [ABP CLI](CLI.md) to install this package.

### ### Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.BackgroundWorkers.Quartz
````
```

### ### Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.BackgroundWorkers.Quartz](<https://www.nuget.org/packages/Volo.Abp.BackgroundWorkers.Quartz>) NuGet package to your project:

```
```
Install-Package Volo.Abp.BackgroundWorkers.Quartz
````
```

2. Add the `AbpBackgroundWorkersQuartzModule` to the dependency list of your module:

```
```csharp
[DependsOn(
    //...other dependencies
    typeof(AbpBackgroundWorkersQuartzModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
````
```

> Quartz background worker integration provided `QuartzPeriodicBackgroundWorkerAdapter` to adapt `PeriodicBackgroundWorkerBase` and `AsyncPeriodicBackgroundWorkerBase` derived class. So, you can still follow the [background workers document](Background-Workers.md) to define the background worker.

### ## Configuration

See [Configuration](Background-Jobs-Quartz#Configuration).

### ## Create a Background Worker

A background work is a class that derives from the `QuartzBackgroundWorkerBase` base class. for example. A simple worker class is shown below:

```
```csharp
public class MyLogWorker : QuartzBackgroundWorkerBase
{
    public MyLogWorker()
    {
````
```

```

 JobDetail =
JobBuilder.Create<MyLogWorker>().WithIdentity(nameof(MyLogWorker)).Build();
 Trigger =
TriggerBuilder.Create().WithIdentity(nameof(MyLogWorker)).StartNow().Build();
 }

 public override Task Execute(IJobExecutionContext context)
 {
 Logger.LogInformation("Executed MyLogWorker..!");
 return Task.CompletedTask;
 }
}
...

```

We simply implemented the `Execute` method to write a log. The background worker is a \*\*singleton by default\*\*. If you want, you can also implement a [dependency interface](Dependency-Injection#DependencyInterfaces) to register it as another life cycle.

> Tips: Add identity to background workers is a best practice, because quartz distinguishes different jobs based on identity.

#### ## Add to BackgroundWorkerManager

Default background workers are \*\*automatically\*\* added to the `BackgroundWorkerManager` when the application is \*\*initialized\*\*. You can set '`AutoRegister`' property value to '`false`', if you want to add it manually:

```

```csharp
public class MyLogWorker : QuartzBackgroundWorkerBase
{
    public MyLogWorker()
    {
        AutoRegister = false;
        JobDetail =
JobBuilder.Create<MyLogWorker>().WithIdentity(nameof(MyLogWorker)).Build();
        Trigger =
TriggerBuilder.Create().WithIdentity(nameof(MyLogWorker)).StartNow().Build();
    }

    public override Task Execute(IJobExecutionContext context)
    {
        Logger.LogInformation("Executed MyLogWorker..!");
        return Task.CompletedTask;
    }
}
...

```

If you want to globally disable auto add worker, you can global disable via '`AbpBackgroundWorkerQuartzOptions`' options:

```

```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundWorkersQuartzModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{

```

```

 public override void ConfigureServices(ServiceConfigurationContext
context)
{
 Configure<AbpBackgroundWorkerQuartzOptions>(options =>
{
 options.IsAutoRegisterEnabled = false;
});
}
}..
```
## Advanced topics
```

Customize ScheduleJob

Assume you have a worker executes every 10 minutes, but because server is unavailable for 30 minutes, 3 executions are missed. You want to execute all missed times after the server is available. You should define your background worker like this:

```

```csharp
public class MyLogWorker : QuartzBackgroundWorkerBase
{
 public MyLogWorker()
 {
 JobDetail =
JobBuilder.Create<MyLogWorker>().WithIdentity(nameof(MyLogWorker)).Build();
 Trigger =
TriggerBuilder.Create().WithIdentity(nameof(MyLogWorker)).WithSimpleSchedule(
s=>s.WithIntervalInMinutes(1).RepeatForever().WithMisfireHandlingInstructionI
gnoreMisfires()).Build();

 ScheduleJob = async scheduler =>
 {
 if (!await scheduler.CheckExists(JobDetail.Key))
 {
 await scheduler.ScheduleJob(JobDetail, Trigger);
 }
 };
}

 public override Task Execute(IJobExecutionContext context)
 {
 Logger.LogInformation("Executed MyLogWorker..!");
 return Task.CompletedTask;
 }
}..
```

```

In the example we defined the worker execution interval to be 10 minutes and set 'WithMisfireHandlingInstructionIgnoreMisfires'. we customized 'ScheduleJob' and add worker to quartz only when the background worker does not exist.

More

Please see Quartz's [documentation](<https://www.quartz-scheduler.net/documentation/index.html>) for more information.

7.3.2 Hangfire Integration

```
# Hangfire Background Worker Manager
```

[[Hangfire](https://www.hangfire.io/)](https://www.hangfire.io/) is an advanced background jobs and worker manager. You can integrate Hangfire with the ABP Framework to use it instead of the [[default background worker manager](#)](Background-Workers.md).

The major advantage is that you can use the same server farm to manage your Background Jobs and Workers, as well as leverage the advanced scheduling that is available from Hangfire for [[Recurring Jobs](#)](https://docs.hangfire.io/en/latest/background-methods/performing-recurrent-tasks.html?highlight=recurring), aka Background Workers.

```
## Installation
```

It is suggested to use the [[ABP CLI](#)](CLI.md) to install this package.

```
### Using the ABP CLI
```

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.BackgroundWorkers.Hangfire
````
```

```
### Manual Installation
```

If you want to manually install;

1. Add the [[Volo.Abp.BackgroundWorkers.Hangfire](#)](https://www.nuget.org/packages/Volo.Abp.BackgroundWorkers.Hangfire) NuGet package to your project:

```
```
Install-Package Volo.Abp.BackgroundWorkers.Hangfire
````
```

2. Add the `AbpBackgroundWorkersHangfireModule` to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpBackgroundWorkersHangfireModule) //Add the new module
 dependency
)]
public class YourModule : AbpModule
{
}
````
```

> Hangfire background worker integration provides an adapter `HangfirePeriodicBackgroundWorkerAdapter` to automatically load any

`PeriodicBackgroundWorkerBase` and `AsyncPeriodicBackgroundWorkerBase` derived classes as `IHangfireBackgroundWorker` instances. This allows you to still to easily switch over to use Hangfire as the background manager even you have existing background workers that are based on the [default background workers implementation](Background-Workers.md).

Configuration

You can install any storage for Hangfire. The most common one is SQL Server (see the [Hangfire.SqlServer](<https://www.nuget.org/packages/Hangfire.SqlServer>) NuGet package).

After you have installed these NuGet packages, you need to configure your project to use Hangfire.

1. First, we change the `Module` class (example: `<YourProjectName>HttpApiHostModule`) to add Hangfire configuration of the storage and connection string in the `ConfigureServices` method:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 var configuration = context.Services.GetConfiguration();
 var hostingEnvironment = context.Services.GetHostingEnvironment();

 //... other configurations.

 ConfigureHangfire(context, configuration);
}

private void ConfigureHangfire(ServiceConfigurationContext context,
IConfiguration configuration)
{
 context.Services.AddHangfire(config =>
 {

config.UseSqlServerStorage(configuration.GetConnectionString("Default"));
 });
}
...```

```

> You have to configure a storage for Hangfire.

2. If you want to use hangfire's dashboard, you can add `UseAbpHangfireDashboard` call in the `OnApplicationInitialization` method in `Module` class

```
```csharp
public override void
OnApplicationInitialization(ApplicationInitializationContext context)
{
    var app = context.GetApplicationBuilder();

    // ... others

    app.UseAbpHangfireDashboard(); //should add to the request pipeline
before the app.UseConfiguredEndpoints()```

```

```

        app.UseConfiguredEndpoints();
    } ...
```
Create a Background Worker

`'HangfireBackgroundWorkerBase'` is an easy way to create a background worker.

```csharp
public class MyLogWorker : HangfireBackgroundWorkerBase
{
    public MyLogWorker()
    {
        RecurringJobId = nameof(MyLogWorker);
        CronExpression = Cron.Daily();
    }

    public override Task DoWorkAsync(CancellationToken cancellationToken =
default)
    {
        Logger.LogInformation("Executed MyLogWorker..!");
        return Task.CompletedTask;
    }
} ...
```

```

\* **RecurringJobId** Is an optional parameter, see [[Hangfire document](#)](<https://docs.hangfire.io/en/latest/background-methods/performing-recurrent-tasks.html>)

\* **CronExpression** Is a CRON expression, see [[CRON expression](#)]([https://en.wikipedia.org/wiki/Cron#CRON\\_expression](https://en.wikipedia.org/wiki/Cron#CRON_expression))

> You can directly implement the `IHangfireBackgroundWorker`, but `HangfireBackgroundWorkerBase` provides some useful properties like Logger.

### ### UnitOfWork

```

```csharp
public class MyLogWorker : HangfireBackgroundWorkerBase, IMyLogWorker
{
    public MyLogWorker()
    {
        RecurringJobId = nameof(MyLogWorker);
        CronExpression = Cron.Daily();
    }

    public override Task DoWorkAsync(CancellationToken cancellationToken =
default)
    {
        using (var uow =
LazyServiceProvider.LazyGetRequiredService<IUnitOfWorkManager>().Begin())
        {
            Logger.LogInformation("Executed MyLogWorker..!");
            return Task.CompletedTask;
        }
    }
} ...
```

```

```
Register BackgroundWorkerManager
```

After creating a background worker class, you should add it to the `IBackgroundWorkerManager`. The most common place is the `OnApplicationInitializationAsync` method of your module class:

```
``` csharp
[DependsOn(typeof(AbpBackgroundWorkersModule))]
public class MyModule : AbpModule
{
    public override async Task OnApplicationInitializationAsync(
        ApplicationInitializationContext context)
    {
        await context.AddBackgroundWorkerAsync<MyLogWorker>();
    }
}...
```

`context.AddBackgroundWorkerAsync(...)` is a shortcut extension method for the expression below:

```
``` csharp
context.ServiceProvider
 .GetRequiredService<IBackgroundWorkerManager>()
 .AddAsync(
 context
 .ServiceProvider
 .GetRequiredService<MyLogWorker>()
);
...;
```

So, it resolves the given background worker and adds to the `IBackgroundWorkerManager`.

While we generally add workers in `OnApplicationInitializationAsync`, there are no restrictions on that. You can inject `IBackgroundWorkerManager` anywhere and add workers at runtime. Background worker manager will stop and release all the registered workers when your application is being shut down.

## 7.4 BLOB Storing

### 7.4.1 BLOB Storing System

#### # BLOB Storing

It is typical to \*\*store file contents\*\* in an application and read these file contents on need. Not only files, but you may also need to save various types of \*\*large binary objects\*\*, a.k.a. [BLOB]([https://en.wikipedia.org/wiki/Binary\\_large\\_object](https://en.wikipedia.org/wiki/Binary_large_object))s, into a \*\*storage\*\*. For example, you may want to save user profile pictures.

A BLOB is a typically \*\*byte array\*\*. There are various places to store a BLOB item; storing in the local file system, in a shared database or on the [Azure BLOB storage](<https://azure.microsoft.com/en-us/services/storage/blobs/>) can be options.

The ABP Framework provides an abstraction to work with BLOBs and provides some pre-built storage providers that you can easily integrate to. Having such an abstraction has some benefits;

- \* You can \*\*easily integrate\*\* to your favorite BLOB storage providers with a few lines of configuration.
- \* You can then \*\*easily change\*\* your BLOB storage without changing your application code.
- \* If you want to create \*\*reusable application modules\*\*, you don't need to make assumption about how the BLOBs are stored.

ABP BLOB Storage system is also compatible to other ABP Framework features like [\[multi-tenancy\]](#)(Multi-Tenancy.md).

## ## BLOB Storage Providers

The ABP Framework has already the following storage provider implementations:

- \* [\[File System\]](#)(Blob-Storing-File-System.md): Stores BLOBs in a folder of the local file system, as standard files.
- \* [\[Database\]](#)(Blob-Storing-Database.md): Stores BLOBs in a database.
- \* [\[Azure\]](#)(Blob-Storing-Azure.md): Stores BLOBs on the [\[Azure BLOB storage\]](#)(<https://azure.microsoft.com/en-us/services/storage/blobs/>).
- \* [\[Aliyun\]](#)(Blob-Storing-Aliyun.md): Stores BLOBs on the [\[Aliyun Storage Service\]](#)(<https://help.aliyun.com/product/31815.html>).
- \* [\[Minio\]](#)(Blob-Storing-Minio.md): Stores BLOBs on the [\[MinIO Object storage\]](#)(<https://min.io/>).
- \* [\[Aws\]](#)(Blob-Storing-Aws.md): Stores BLOBs on the [\[Amazon Simple Storage Service\]](#)(<https://aws.amazon.com/s3/>).

More providers will be implemented by the time. You can [\[request\]](#)(<https://github.com/abpframework/abp/issues/new>) it for your favorite provider or [\[create it yourself\]](#)(Blob-Storing-Custom-Provider.md) and [\[contribute\]](#)(Contribution/Index.md) to the ABP Framework.

Multiple providers \*\*can be used together\*\* by the help of the \*\*container system\*\*, where each container can uses a different provider.

> BLOB storing system can not work unless you \*\*configure a storage provider\*\*. Refer to the linked documents for the storage provider configurations.

## ## Installation

[\[Volo.Abp.BlobStoring\]](#)(<https://www.nuget.org/packages/Volo.Abp.BlobStoring>) is the main package that defines the BLOB storing services. You can use this package to use the BLOB Storing system without depending a specific storage provider.

Use the ABP CLI to add this package to your project:

- \* Install the [\[ABP CLI\]](#)(<https://docs.abp.io/en/abp/latest/CLI>), if you haven't installed it.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `'Volo.Abp.BlobStoring'` package.
- \* Run `'abp add-package Volo.Abp.BlobStoring'` command.

If you want to do it manually, install the [Volo.Abp.BlobStoring](<https://www.nuget.org/packages/Volo.Abp.BlobStoring>) NuGet package to your project and add `'[DependsOn(typeof(AbpBlobStoringModule))]'` to the [ABP module](Module-Development-Basics.md) class inside your project.

### ## The IBlobContainer

'IBlobContainer' is the main interface to store and read BLOBs. Your application may have multiple containers and each container can be separately configured. But, there is a \*\*default container\*\* that can be simply used by [injecting](Dependency-Injection.md) the 'IBlobContainer'.

\*\*Example: Simply save and read bytes of a named BLOB\*\*

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.BlobStoring;
using Volo.Abp.DependencyInjection;

namespace AbpDemo
{
    public class MyService : ITransientDependency
    {
        private readonly IBlobContainer _blobContainer;

        public MyService(IBlobContainer blobContainer)
        {
            _blobContainer = blobContainer;
        }

        public async Task SaveBytesAsync(byte[] bytes)
        {
            await _blobContainer.SaveAsync("my-blob-1", bytes);
        }

        public async Task<byte[]> GetBytesAsync()
        {
            return await _blobContainer.GetAllBytesOrNullAsync("my-blob-1");
        }
    }
}...```

```

This service saves the given bytes with the 'my-blob-1' name and then gets the previously saved bytes with the same name.

> A BLOB is a named object and **each BLOB should have a unique name**, which is an arbitrary string.

'IBlobContainer' can work with 'Stream' and 'byte[]' objects, which will be detailed in the next sections.

Saving BLOBs

'SaveAsync' method is used to save a new BLOB or replace an existing BLOB. It can save a 'Stream' by default, but there is a shortcut extension method to save byte arrays.

`'SaveAsync'` gets the following parameters:

- * `**name**` (`string`): Unique name of the BLOB.
- * `**stream**` (`Stream`) or `**bytes**` (`byte[]`): The stream to read the BLOB content or a byte array.
- * `**overrideExisting**` (`bool`): Set `'true'` to replace the BLOB content if it does already exists. Default value is `'false'` and throws `'BlobAlreadyExistsException'` if there is already a BLOB in the container with the same name.

Reading/Getting BLOBS

- * `'GetAsync'`: Only gets a BLOB name and returns a `'Stream'` object that can be used to read the BLOB content. Always `**dispose the stream**` after using it. This method throws exception, if it can not find the BLOB with the given name.
- * `'GetOrNullAsync'`: In opposite to the `'GetAsync'` method, this one returns `'null'` if there is no BLOB found with the given name.
- * `'GetAllBytesAsync'`: Returns a `'byte[]'` instead of a `'Stream'`. Still throws exception if can not find the BLOB with the given name.
- * `'GetAllBytesOrNullAsync'`: In opposite to the `'GetAllBytesAsync'` method, this one returns `'null'` if there is no BLOB found with the given name.

Deleting BLOBS

`'DeleteAsync'` method gets a BLOB name and deletes the BLOB data. It doesn't throw any exception if given BLOB was not found. Instead, it returns a `'bool'` indicating that the BLOB was actually deleted or not, if you care about it.

Other Methods

- * `'ExistsAsync'` method simply checks if there is a BLOB in the container with the given name.

About Naming the BLOBS

There is not a rule for naming the BLOBS. A BLOB name is just a string that is unique per container (and per tenant – see the "**Multi-Tenancy**" section). However, different storage providers may conventionally implement some practices. For example, the [File System Provider](Blob-Storing-File-System.md) use directory separators ('/') and file extensions in your BLOB name (if your BLOB name is `'images/common/x.png'` then it is saved as `'x.png'` in the `'images/common'` folder inside the root container folder).

Typed IBlobContainer

Typed BLOB container system is a way of creating and managing `**multiple containers**` in an application;

- * `**Each container is separately stored**`. That means the BLOB names should be unique in a container and two BLOBS with the same name can live in different containers without effecting each other.
- * `**Each container can be separately configured**`, so each container can use a different storage provider based on your configuration.

To create a typed container, you need to create a simple class decorated with the `'BlobContainerName'` attribute:

```
```csharp
using Volo.Abp.BlobStoring;

namespace AbpDemo
{
 [BlobContainerName("profile-pictures")]
 public class ProfilePictureContainer
 {

 }
}
```

```

> If you don't use the `BlobContainerName` attribute, ABP Framework uses the full name of the class (with namespace), but it is always recommended to use a container name which is stable and does not change even if you rename the class.

Once you create the container class, you can inject `IBlobContainer<T>` for your container type.

****Example: An [application service](Application-Services.md) to save and read profile picture of the [current user](CurrentUser.md)****

```
```csharp
[Authorize]
public class ProfileAppService : ApplicationService
{
 private readonly IBlobContainer<ProfilePictureContainer> _blobContainer;

 public ProfileAppService(IBlobContainer<ProfilePictureContainer>
blobContainer)
 {
 _blobContainer = blobContainer;
 }

 public async Task SaveProfilePictureAsync(byte[] bytes)
 {
 var blobName = CurrentUser.GetId().ToString();
 await _blobContainer.SaveAsync(blobName, bytes);
 }

 public async Task<byte[]> GetProfilePictureAsync()
 {
 var blobName = CurrentUser.GetId().ToString();
 return await _blobContainer.GetAllBytesOrNullAsync(blobName);
 }
}
```

```

`IBlobContainer<T>` has the same methods with the `IBlobContainer`.

> It is a good practice to **always use a typed container while developing re-usable modules**, so the final application can configure the provider for your container without effecting the other containers.

The Default Container

If you don't use the generic argument and directly inject the `IBlobContainer` (as explained before), you get the default container. Another way of injecting the default container is using `IBlobContainer<DefaultContainer>`, which returns exactly the same container.

The name of the default container is 'default'.

Named Containers

Typed containers are just shortcuts for named containers. You can inject and use the `IBlobContainerFactory` to get a BLOB container by its name:

```
```csharp
public class ProfileAppService : ApplicationService
{
 private readonly IBlobContainer _blobContainer;

 public ProfileAppService(IBlobContainerFactory blobContainerFactory)
 {
 _blobContainer = blobContainerFactory.Create("profile-pictures");
 }

 //...
}
```

```

IBlobContainerFactory

`IBlobContainerFactory` is the service that is used to create the BLOB containers. One example was shown above.

Example: Create a container by name

```
```csharp
var blobContainer = blobContainerFactory.Create("profile-pictures");
```

```

Example: Create a container by type

```
```csharp
var blobContainer = blobContainerFactory.Create<ProfilePictureContainer>();
```

```

> You generally don't need to use the `IBlobContainerFactory` since it is used internally, when you inject a `IBlobContainer` or `IBlobContainer<T>`.

Configuring the Containers

Containers should be configured before using them. The most fundamental configuration is to **select a BLOB storage provider** (see the "**BLOB Storage Providers**" section above).

`AbpBlobStoringOptions` is the [options class](Options.md) to configure the containers. You can configure the options inside the `ConfigureServices` method of your [module](Module-Development-Basics.md).

Configure a Single Container

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.Configure<ProfilePictureContainer>(container =>
 {
 //TODO...
 });
});
```

```

This example configures the `ProfilePictureContainer`. You can also configure by the container name:

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.Configure("profile-pictures", container =>
 {
 //TODO...
 });
});
```

```

Configure the Default Container

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.ConfigureDefault(container =>
 {
 //TODO...
 });
});
```

```

> There is a special case about the default container; If you don't specify a configuration for a container, it **fallbacks to the default container configuration**. This is a good way to configure defaults for all containers and specialize configuration for a specific container when needed.

Configure All Containers

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.ConfigureAll((containerName, containerConfiguration)
=>
 {
 //TODO...
 });
});
```

```

This is a way to configure all the containers.

> The main difference from configuring the default container is that `'ConfigureAll'` overrides the configuration even if it was specialized for a specific container.

Multi-Tenancy

If your application is set as multi-tenant, the BLOB Storage system **works seamlessly with the [multi-tenancy](Multi-Tenancy.md)**. All the providers implement multi-tenancy as a standard feature. They **isolate BLOBs** of different tenants from each other, so they can only access to their own BLOBs. It means you can use the **same BLOB name for different tenants**.

If your application is multi-tenant, you may want to control **multi-tenancy behavior** of the containers individually. For example, you may want to **disable multi-tenancy** for a specific container, so the BLOBs inside it will be **available to all the tenants**. This is a way to share BLOBs among all tenants.

Example: Disable multi-tenancy for a specific container

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.Configure<ProfilePictureContainer>(container =>
 {
 container.IsMultiTenant = false;
 });
});```

```

> If your application is not multi-tenant, no worry, it works as expected. You don't need to configure the `'IsMultiTenant'` option.

## ## Extending the BLOB Storing System

Most of the times, you won't need to customize the BLOB storage system except [[creating a custom BLOB storage provider](#)](Blob-Storing-Custom-Provider.md). However, you can replace any service (injected via [[dependency injection](#)](Dependency-Injection.md)), if you need. Here, some other services not mentioned above, but you may want to know:

- \* `'IBlobProviderSelector'` is used to get a `'IBlobProvider'` instance by a container name. Default implementation (`'DefaultBlobProviderSelector'`) selects the provider using the configuration.
- \* `'IBlobContainerConfigurationProvider'` is used to get the `'BlobContainerConfiguration'` for a given container name. Default implementation (`'DefaultBlobContainerConfigurationProvider'`) gets the configuration from the `'AbpBlobStoringOptions'` explained above.

## ## BLOB Storing vs File Management System

Notice that BLOB storing is not a file management system. It is a low level system that is used to save, get and delete named BLOBs. It doesn't provide a hierarchical structure like directories, you may expect from a typical file system.

If you want to create folders and move files between folders, assign permissions to files and share files between users then you need to implement your own application on top of the BLOB Storage system.

## ## See Also

- \* [Creating a custom BLOB storage provider](Blob-Storing-Custom-Provider.md)

### 7.4.2 Storage Providers

#### 7.4.2.1 File System Provider

##### # BLOB Storing File System Provider

File System Storage Provider is used to store BLOBs in the local file system as standard files inside a folder.

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to configure containers to use the file system.

##### ## Installation

Use the ABP CLI to add [Volo.Abp.BlobStoring.FileSystem](<https://www.nuget.org/packages/Volo.Abp.BlobStoring.FileSystem>) NuGet package to your project:

- \* Install the [ABP CLI](<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.BlobStoring.FileSystem` package.
- \* Run `abp add-package Volo.Abp.BlobStoring.FileSystem` command.

If you want to do it manually, install the [Volo.Abp.BlobStoring.FileSystem](<https://www.nuget.org/packages/Volo.Abp.BlobStoring.FileSystem>) NuGet package to your project and add `[DependsOn(typeof(AbpBlobStoringFileSystemModule))]` to the [ABP module](Module-Development-Basics.md) class inside your project.

##### ## Configuration

Configuration is done in the `ConfigureServices` method of your [module](Module-Development-Basics.md) class, as explained in the [BLOB Storing document](Blob-Storing.md).

\*\*Example: Configure to use the File System storage provider by default\*\*

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
    options.Containers.ConfigureDefault(container =>
    {
        container.UseFileSystem(fileSystem =>
        {
            fileSystem.BasePath = "C:\\my-files";
        });
    });
}
```

```
});  
});
```

'UseFileSystem' extension method is used to set the File System Provider for a container and configure the file system options.

> See the [BLOB Storing document](Blob-Storing.md) to learn how to configure this provider for a specific container.

Options

- * **BasePath** (string): The base folder path to store BLOBS. It is required to set this option.
- * **AppendContainerNameToBasePath** (bool; default: `true`): Indicates whether to create a folder with the container name inside the base folder. If you store multiple containers in the same 'BaseFolder', leave this as 'true'. Otherwise, you can set it to 'false' if you don't like an unnecessarily deeper folder hierarchy.

File Path Calculation

File System Provider organizes BLOB files inside folders and implements some conventions. The full path of a BLOB file is determined by the following rules by default:

- * It starts with the 'BasePath' configured as shown above.
- * Appends 'host' folder if [current tenant](Multi-Tenancy.md) is 'null' (or multi-tenancy is disabled for the container - see the [BLOB Storing document](Blob-Storing.md) to learn how to disable multi-tenancy for a container).
- * Appends 'tenants/<tenant-id>' folder if current tenant is not 'null'.
- * Appends the container's name if 'AppendContainerNameToBasePath' is 'true'. If container name contains '/', this will result with nested folders.
- * Appends the BLOB name. If the BLOB name contains '/' it creates folders. If the BLOB name contains '.' it will have a file extension.

Extending the File System BLOB Provider

- * 'FileSystemBlobProvider' is the main service that implements the File System storage. You can inherit from this class and [override](Customizing-Application-Modules-Overriding-Services.md) methods to customize it.
- * The 'IBlobFilePathCalculator' service is used to calculate the file paths. Default implementation is the 'DefaultBlobFilePathCalculator'. You can replace/override it if you want to customize the file path calculation.

7.4.2.2 Database Provider

BLOB Storing Database Provider

BLOB Storing Database Storage Provider can store BLOBS in a relational or non-relational database.

There are two database providers implemented;

```
* [Volo.Abp.BlobStoring.Database.EntityFrameworkCore](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Database.EntityFrameworkCore) package implements for [EF Core](Entity-Framework-Core.md), so it can store BLOBs in [any DBMS supported](https://docs.microsoft.com/en-us/ef/core/providers/) by the EF Core.  
* [Volo.Abp.BlobStoring.Database.MongoDB](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Database.MongoDB) package implements for [MongoDB](MongoDB.md).
```

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to configure containers to use a database as the storage provider.

Installation

Automatic Installation

If you've created your solution based on the [application startup template](Startup-Templates/Application.md), you can use the `abp add-module` [CLI](CLI.md) command to automatically add related packages to your solution.

Open a command prompt (terminal) in the folder containing your solution (`.sln`) file and run the following command:

```
```bash
abp add-module Volo.Abp.BlobStoring.Database
````
```

This command adds all the NuGet packages to corresponding layers of your solution. If you are using EF Core, it adds necessary configuration, adds a new database migration and updates the database.

Manual Installation

Here, all the NuGet packages defined by this provider;

```
* [Volo.Abp.BlobStoring.Database.Domain.Shared](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Database.Domain.Shared)  
* [Volo.Abp.BlobStoring.Database.Domain](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Database.Domain)  
* [Volo.Abp.BlobStoring.Database.EntityFrameworkCore](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Database.EntityFrameworkCore)  
* [Volo.Abp.BlobStoring.Database.MongoDB](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Database.MongoDB)
```

You can only install `Volo.Abp.BlobStoring.Database.EntityFrameworkCore` or `Volo.Abp.BlobStoring.Database.MongoDB` (based on your preference) since they depends on the other packages.

After installation, add `'DepenedsOn'` attribute to your related [module](Module-Development-Basics.md). Here, the list of module classes defined by the related NuGet packages listed above:

```
* `BlobStoringDatabaseDomainModule`  
* `BlobStoringDatabaseDomainSharedModule`  
* `BlobStoringDatabaseEntityFrameworkCoreModule`  
* `BlobStoringDatabaseMongoDbModule`
```

Whenever you add a NuGet package to a project, also add the module class dependency.

If you are using EF Core, you also need to configure your `DbContext` to add BLOB storage tables to your database schema. Call `'builder.ConfigureBlobStoring()'` extension method inside the `'OnModelCreating'` method to include mappings to your `DbContext`. Then you can use the standard `'Add-Migration'` and `'Update-Database'` [commands](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>) to create necessary tables in your database.

Configuration

Connection String

If you will use your `'Default'` connection string, you don't need to any additional configuration.

If you want to use a separate database for BLOB storage, use the `'AbpBlobStoring'` as the [connection string](Connection-Strings.md) name in your configuration file (`'appsettings.json'`). In this case, also read the [EF Core Migrations](Entity-Framework-Core-Migrations.md) document to learn how to create and use a different database for a desired module.

Configuring the Containers

If you are using only the database storage provider, you don't need to manually configure it, since it is automatically done. If you are using multiple storage providers, you may want to configure it.

Configuration is done in the `'ConfigureServices'` method of your `[module](Module-Development-Basics.md)` class, as explained in the [BLOB Storing document](Blob-Storing.md).

Example: Configure to use the database storage provider by default

```
```csharp  
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.ConfigureDefault(container =>
 {
 container.UseDatabase();
 });
});
```
```

> See the [BLOB Storing document](Blob-Storing.md) to learn how to configure this provider for a specific container.

Additional Information

It is expected to use the [BLOB Storing services](Blob-Storing.md) to use the BLOB storing system. However, if you want to work on the database tables/entities, you can use the following information.

Entities

Entities defined for this module:

- * `DatabaseBlobContainer` (aggregate root) represents a container stored in the database.
- * `DatabaseBlob` (aggregate root) represents a BLOB in the database.

See the [entities document](Entities.md) to learn what is an entity and aggregate root.

Repositories

- * `IDatabaseBlobContainerRepository`
- * `IDatabaseBlobRepository`

You can also use ` IRepository<DatabaseBlobContainer, Guid>` and ` IRepository<DatabaseBlob, Guid>` to take the power of IQueryables. See the [repository document](Repositories.md) for more.

Other Services

- * `DatabaseBlobProvider` is the main service that implements the database BLOB storage provider, if you want to override/replace it via [dependency injection](Dependency-Injection.md) (don't replace `IBlobProvider` interface, but replace `DatabaseBlobProvider` class).

7.4.2.3 Azure Provider

BLOB Storing Azure Provider

BLOB Storing Azure Provider can store BLOBs in [Azure Blob storage](https://azure.microsoft.com/en-us/services/storage/blobs/).

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to configure containers to use a Azure BLOB as the storage provider.

Installation

Use the ABP CLI to add [Volo.Abp.BlobStoring.Azure](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Azure) NuGet package to your project:

- * Install the [ABP CLI](https://docs.abp.io/en/abp/latest/CLI) if you haven't installed before.
- * Open a command line (terminal) in the directory of the `*.csproj` file you want to add the `Volo.Abp.BlobStoring.Azure` package.
- * Run `abp add-package Volo.Abp.BlobStoring.Azure` command.

If you want to do it manually, install the [Volo.Abp.BlobStoring.Azure](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Azure) NuGet package to your project and add

`[DependsOn(typeof(AbpBlobStoringAzureModule))]` to the [ABP module](Module-Development-Basics.md) class inside your project.

Configuration

Configuration is done in the `ConfigureServices` method of your [module](Module-Development-Basics.md) class, as explained in the [BLOB Storing document](Blob-Storing.md).

Example: Configure to use the azure storage provider by default

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 optionsContainers.ConfigureDefault(container =>
 {
 container.UseAzure(azure =>
 {
 azure.ConnectionString = "your azure connection string";
 azure.ContainerName = "your azure container name";
 azure.CreateContainerIfNotExists = true;
 });
 });
});```

```

> See the [BLOB Storing document](Blob-Storing.md) to learn how to configure this provider for a specific container.

## ### Options

- \* \*\*ConnectionString\*\* (string): A connection string includes the authorization information required for your application to access data in an Azure Storage account at runtime using Shared Key authorization. Please refer to Azure documentation: <https://docs.microsoft.com/en-us/azure/storage/common/storage-configure-connection-string>
- \* \*\*ContainerName\*\* (string): You can specify the container name in azure. If this is not specified, it uses the name of the BLOB container defined with the `BlobContainerName` attribute (see the [BLOB storing document](Blob-Storing.md)). Please note that Azure has some \*\*rules for naming containers\*\*. A container name must be a valid DNS name, conforming to the [following naming rules](https://docs.microsoft.com/en-us/rest/api/storageservices/naming-and-referencing-containers--blobs--and-metadata#container-names):
  - \* Container names must start or end with a letter or number, and can contain only letters, numbers, and the dash (-) character.
  - \* Every dash (-) character must be immediately preceded and followed by a letter or number; consecutive dashes are not permitted in container names.
  - \* All letters in a container name must be \*\*lowercase\*\*.
  - \* Container names must be from \*\*3\*\* through \*\*63\*\* characters long.
- \* \*\*CreateContainerIfNotExists\*\* (bool): Default value is `false`, If a container does not exist in azure, `AzureBlobProvider` will try to create it.

## ## Azure Blob Name Calculator

Azure Blob Provider organizes BLOB name and implements some conventions. The full name of a BLOB is determined by the following rules by default:

```
* Appends 'host' string if [current tenant](Multi-Tenancy.md) is 'null' (or multi-tenancy is disabled for the container – see the [BLOB Storing document](Blob-Storing.md) to learn how to disable multi-tenancy for a container).
* Appends 'tenants/<tenant-id>' string if current tenant is not 'null'.
* Appends the BLOB name.
```

#### ## Other Services

```
* `AzureBlobProvider` is the main service that implements the Azure BLOB storage provider, if you want to override/replace it via [dependency injection](Dependency-Injection.md) (don't replace `IBlobProvider` interface, but replace `AzureBlobProvider` class).
* `IAzureBlobNameCalculator` is used to calculate the full BLOB name (that is explained above). It is implemented by the `DefaultAzureBlobNameCalculator` by default.
```

#### 7.4.2.4 Aliyun Provider

##### # BLOB Storing Aliyun Provider

BLOB Storing Aliyun Provider can store BLOBs in [Aliyun Blob storage](https://help.aliyun.com/product/31815.html).

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to configure containers to use a Aliyun BLOB as the storage provider.

##### ## Installation

Use the ABP CLI to add [Volo.Abp.BlobStoring.Aliyun](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Aliyun) NuGet package to your project:

```
* Install the [ABP CLI](https://docs.abp.io/en/abp/latest/CLI) if you haven't installed before.
* Open a command line (terminal) in the directory of the `*.csproj` file you want to add the `Volo.Abp.BlobStoring.Aliyun` package.
* Run `abp add-package Volo.Abp.BlobStoring.Aliyun` command.
```

If you want to do it manually, install the [Volo.Abp.BlobStoring.Aliyun](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Aliyun) NuGet package to your project and add `'[DependsOn(typeof(AbpBlobStoringAliyunModule))]'` to the [ABP module](Module-Development-Basics.md) class inside your project.

##### ## Configuration

Configuration is done in the `ConfigureServices` method of your [module](Module-Development-Basics.md) class, as explained in the [BLOB Storing document](Blob-Storing.md).

\*\*Example: Configure to use the Aliyun storage provider by default\*\*

```
```csharp
```

```

Configure<AbpBlobStoringOptions>(options =>
{
    options.Containers.ConfigureDefault(container =>
    {
        container.UseAliyun(aliyun =>
        {
            aliyun.AccessKeyId = "your aliyun access key id";
            aliyun.AccessKeySecret = "your aliyun access key secret";
            aliyun.Endpoint = "your oss endpoint";
            aliyun.RegionId = "your sts region id";
            aliyun.RoleArn = "the arn of ram role";
            aliyun.RoleSessionName = "the name of the certificate";
            aliyun.Policy = "policy";
            aliyun.DurationSeconds = "expiration date";
            aliyun.ContainerName = "your aliyun container name";
            aliyun.CreateContainerIfNotExists = true;
        });
    });
});
...

```

> See the [BLOB Storing document](Blob-Storing.md) to learn how to configure this provider for a specific container.

Options

- * **AccessKeyId** ([NotNull]string): AccessKey is the key to access the Alibaba Cloud API. It has full permissions for the account. Please keep it safe! Recommend to follow [Alibaba Cloud security best practices](https://help.aliyun.com/document_detail/102600.html), Use RAM sub-user AccessKey to call API.
- * **AccessKeySecret** ([NotNull]string): Same as above.
- * **Endpoint** ([NotNull]string): Endpoint is the external domain name of OSS. See the [document](https://help.aliyun.com/document_detail/31837.html) for details.
- * **UseSecurityTokenService** (bool): Use [STS temporary credentials](https://help.aliyun.com/document_detail/100624.html) to access OSS services, default: `false`.
- * **RegionId** (string): Access address of STS service. See the [document](https://help.aliyun.com/document_detail/66053.html) for details.
- * **RoleArn** ([NotNull]string): STS required role ARN. See the [document](https://help.aliyun.com/document_detail/100624.html) for details.
- * **RoleSessionName** ([NotNull]string): Used to identify the temporary access credentials, it is recommended to use different application users to distinguish.
- * **Policy** (string): Additional permission restrictions. See the [document](https://help.aliyun.com/document_detail/100680.html) for details.
- * **DurationSeconds** (int): Validity period(s) of a temporary access certificate, minimum is 900 and the maximum is 3600.
- * **ContainerName** (string): You can specify the container name in Aliyun. If this is not specified, it uses the name of the BLOB container defined with the `BlobContainerName` attribute (see the [BLOB storing document](Blob-Storing.md)). Please note that Aliyun has some **rules for naming containers**. A container name must be a valid DNS name, conforming to the [following naming rules](https://help.aliyun.com/knowledge_detail/39668.html):
* Container names must start or end with a letter or number, and can contain only letters, numbers, and the dash (-) character.

- * Container names Must start and end with lowercase letters and numbers.
- * Container names must be from **3** through **63** characters long.
- * **CreateContainerIfNotExists** (bool): Default value is '`false`', If a container does not exist in Aliyun, '`AliyunBlobProvider`' will try to create it.
- * **TemporaryCredentialsCacheKey** (bool): The cache key of STS credentials.

`## Aliyun Blob Name Calculator`

Aliyun Blob Provider organizes BLOB name and implements some conventions. The full name of a BLOB is determined by the following rules by default:

- * Appends '`host`' string if [current tenant](Multi-Tenancy.md) is '`null`' (or multi-tenancy is disabled for the container - see the [BLOB Storing document](Blob-Storing.md) to learn how to disable multi-tenancy for a container).
- * Appends '`tenants/<tenant-id>`' string if current tenant is not '`null`'.
- * Appends the BLOB name.

`## Other Services`

- * '`AliyunBlobProvider`' is the main service that implements the Aliyun BLOB storage provider, if you want to override/replace it via [dependency injection](Dependency-Injection.md) (don't replace '`IBlobProvider`' interface, but replace '`AliyunBlobProvider`' class).
- * '`IAliyunBlobNameCalculator`' is used to calculate the full BLOB name (that is explained above). It is implemented by the '`DefaultAliyunBlobNameCalculator`' by default.
- * '`IOssClientFactory`' is used create OSS client. It is implemented by the '`DefaultOssClientFactory`' by default. You can override/replace it,if you want customize.

`7.4.2.5 Minio Provider`

`# BLOB Storing Minio Provider`

BLOB Storing Minio Provider can store BLOBS in [MinIO Object storage](https://min.io/).

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to configure containers to use a Minio BLOB as the storage provider.

`## Installation`

Use the ABP CLI to add [Volo.Abp.BlobStoring.Minio](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Minio) NuGet package to your project:

- * Install the [ABP CLI](https://docs.abp.io/en/abp/latest/CLI) if you haven't installed before.
- * Open a command line (terminal) in the directory of the '`.csproj`' file you want to add the '`Volo.Abp.BlobStoring.Minio`' package.
- * Run '`abp add-package Volo.Abp.BlobStoring.Minio`' command.

If you want to do it manually, install the [\[Volo.Abp.BlobStoring.Minio\]](https://www.nuget.org/packages/Volo.Abp.BlobStoring.Minio)(<https://www.nuget.org/packages/Volo.Abp.BlobStoring.Minio>) NuGet package to your project and add `'[DependsOn(typeof(AbpBlobStoringMinioModule))]'` to the [\[ABP module\]\(Module-Development-Basics.md\)](#) class inside your project.

Configuration

Configuration is done in the `ConfigureServices` method of your [\[module\]\(Module-Development-Basics.md\)](#) class, as explained in the [\[BLOB Storing document\]\(Blob-Storing.md\)](#).

Example: Configure to use the minio storage provider by default

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.ConfigureDefault(container =>
 {
 container.UseMinio(minio =>
 {
 minio.EndPoint = "your minio endPoint";
 minio.AccessKey = "your minio accessKey";
 minio.SecretKey = "your minio secretKey";
 minio.BucketName = "your minio bucketName";
 });
 });
});```

```

> See the [\[BLOB Storing document\]\(Blob-Storing.md\)](#) to learn how to configure this provider for a specific container.

### ### Options

- \* **\*\*EndPoint\*\*** (string): URL to object storage service. Please refer to MinIO Client SDK for .NET: <https://docs.min.io/docs/dotnet-client-quickstart-guide.html>
- \* **\*\*AccessKey\*\*** (string): Access key is the user ID that uniquely identifies your account.
- \* **\*\*SecretKey\*\*** (string): Secret key is the password to your account.
- \* **\*\*BucketName\*\*** (string): You can specify the bucket name in MinIO. If this is not specified, it uses the name of the BLOB container defined with the **'BlobContainerName'** attribute (see the [\[BLOB storing document\]\(Blob-Storing.md\)](#)). MinIO is the defacto standard for S3 compatibility, So MinIO has some **\*\*rules for naming bucket\*\***. The [\[following rules\]\(https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html\)](#) apply for naming MinIO buckets:
  - \* Bucket names must be between **\*\*3\*\*** and **\*\*63\*\*** characters long.
  - \* Bucket names can consist only of **\*\*lowercase\*\*** letters, numbers, dots (.), and hyphens (-).
  - \* Bucket names must begin and end with a letter or number.
  - \* Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
  - \* Bucket names can't begin with **\*\*xn--\*\*** (for buckets created after February 2020).
  - \* Bucket names must be unique within a partition.

```
* Buckets used with Amazon S3 Transfer Acceleration can't have dots (.) in their names. For more information about transfer acceleration, see Amazon S3 Transfer Acceleration.
* **WithSSL** (bool): Default value is `false`, Chain to MinIO Client object to use https instead of http.
* **CreateContainerIfNotExists** (bool): Default value is `false`, If a bucket does not exist in minio, `MinioBlobProvider` will try to create it.
```

## ## Minio Blob Name Calculator

Minio Blob Provider organizes BLOB name and implements some conventions. The full name of a BLOB is determined by the following rules by default:

- \* Appends `host` string if [current tenant](Multi-Tenancy.md) is `null` (or multi-tenancy is disabled for the container - see the [BLOB Storing document](Blob-Storing.md) to learn how to disable multi-tenancy for a container).
- \* Appends `tenants/<tenant-id>` string if current tenant is not `null`.
- \* Appends the BLOB name.

## ## Other Services

- \* `MinioBlobProvider` is the main service that implements the Minio BLOB storage provider, if you want to override/replace it via [dependency injection](Dependency-Injection.md) (don't replace `IBlobProvider` interface, but replace `MinioBlobProvider` class).
- \* `IMinioBlobNameCalculator` is used to calculate the full BLOB name (that is explained above). It is implemented by the `DefaultMinioBlobNameCalculator` by default.

### 7.4.2.6 AWS Provider

#### # BLOB Storing Aws Provider

BLOB Storing Aws Provider can store BLOBs in [Amazon Simple Storage Service](<https://aws.amazon.com/s3/>).

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to configure containers to use a Aws BLOB as the storage provider.

#### ## Installation

Use the ABP CLI to add [Volo.Abp.BlobStoring.Aws](<https://www.nuget.org/packages/Volo.Abp.BlobStoring.Aws>) NuGet package to your project:

- \* Install the [ABP CLI](<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.BlobStoring.Aws` package.
- \* Run `abp add-package Volo.Abp.BlobStoring.Aws` command.

If you want to do it manually, install the [Volo.Abp.BlobStoring.Aws](<https://www.nuget.org/packages/Volo.Abp.BlobStoring.Aws>)

g.Aws) NuGet package to your project and add  
`[DependsOn(typeof(AbpBlobStoringAwsModule))]' to the [ABP module](Module-Development-Basics.md) class inside your project.

#### ## Configuration

Configuration is done in the `ConfigureServices` method of your [module](Module-Development-Basics.md) class, as explained in the [BLOB Storing document](Blob-Storing.md).

\*\*Example: Configure to use the Aws storage provider by default\*\*

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
    options.Containers.ConfigureDefault(container =>
    {
        container.UseAws(Aws =>
        {
            Aws.AccessKeyId = "your Aws access key id";
            Aws.SecretAccessKey = "your Aws access key secret";
            Aws.UseCredentials = "set true to use credentials";
            Aws.UseTemporaryCredentials = "set true to use temporary
credentials";
            Aws.UseTemporaryFederatedCredentials = "set true to use temporary
federated credentials";
            Aws.ProfileName = "the name of the profile to get credentials
from";
            Aws.ProfilesLocation = "the path to the aws credentials file to
look at";
            Aws.Region = "the system name of the service";
            Aws.Name = "the name of the federated user";
            Aws.Policy = "policy";
            Aws.DurationSeconds = "expiration date";
            Aws.ContainerName = "your Aws container name";
            Aws.CreateContainerIfNotExists = true;
        });
    });
});
```
....
```

> See the [BLOB Storing document](Blob-Storing.md) to learn how to configure this provider for a specific container.

#### ### Options

- \* \*\*AccessKeyId\*\* (string): AWS Access Key ID.
- \* \*\*SecretAccessKey\*\* (string): AWS Secret Access Key.
- \* \*\*UseCredentials\*\* (bool): Use [credentials](<https://docs.aws.amazon.com/AmazonS3/latest/dev/AuthUsingAcctOrUserCredentials.html>) to access AWS services, default : `false`.
- \* \*\*UseTemporaryCredentials\*\* (bool): Use [temporary credentials](<https://docs.aws.amazon.com/AmazonS3/latest/dev/AuthUsingTempSessionToken.html>) to access AWS services, default : `false`.
- \* \*\*UseTemporaryFederatedCredentials\*\* (bool): Use [federated user temporary credentials](<https://docs.aws.amazon.com/AmazonS3/latest/dev/AuthUsingTempFederationToken.html>) to access AWS services, default : `false`.

- \* **\*\*ProfileName\*\*** (string): The [name of the profile](<https://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/net-dg-config-creds.html>) to get credentials from.
- \* **\*\*ProfilesLocation\*\*** (string): The path to the aws credentials file to look at.
- \* **\*\*Region\*\*** (string): The system name of the service.
- \* **\*\*Policy\*\*** (string): An IAM policy in JSON format that you want to use as an inline session policy.
- \* **\*\*DurationSeconds\*\*** (int): Validity period(s) of a temporary access certificate, minimum is 900 and the maximum is 3600. **\*\*note\*\***: Using sub-accounts operated OSS, if the value is 0.
- \* **\*\*ContainerName\*\*** (string): You can specify the container name in Aws. If this is not specified, it uses the name of the BLOB container defined with the **'BlobContainerName'** attribute (see the [BLOB storing document]([Blob-Storing.md](#))). Please note that Aws has some **\*\*rules for naming containers\*\***. A container name must be a valid DNS name, conforming to the [following naming rules](<https://docs.aws.amazon.com/AmazonS3/latest/dev/BucketRestrictions.html>):

  - \* Bucket names must be between **\*\*3\*\*** and **\*\*63\*\*** characters long.
  - \* Bucket names can consist only of **\*\*lowercase\*\*** letters, numbers, dots (.), and hyphens (-).
  - \* Bucket names must begin and end with a letter or number.
  - \* Bucket names must not be formatted as an IP address (for example, 192.168.5.4).
  - \* Bucket names can't begin with **\*\*xn--\*\*** (for buckets created after February 2020).
  - \* Bucket names must be unique within a partition.
  - \* Buckets used with Amazon S3 Transfer Acceleration can't have dots (.) in their names. For more information about transfer acceleration, see Amazon S3 Transfer Acceleration.

- \* **\*\*CreateContainerIfNotExists\*\*** (bool): Default value is '**false**', If a container does not exist in Aws, '**AwsBlobProvider**' will try to create it.

## **## Aws Blob Name Calculator**

Aws Blob Provider organizes BLOB name and implements some conventions. The full name of a BLOB is determined by the following rules by default:

- \* Appends **'host'** string if [current tenant]([Multi-Tenancy.md](#)) is '**null**' (or multi-tenancy is disabled for the container - see the [BLOB Storing document]([Blob-Storing.md](#)) to learn how to disable multi-tenancy for a container).
- \* Appends '**tenants/<tenant-id>**' string if current tenant is not '**null**'.
- \* Appends the BLOB name.

## **## Other Services**

- \* '**AwsBlobProvider**' is the main service that implements the Aws BLOB storage provider, if you want to override/replace it via [dependency injection]([Dependency-Injection.md](#)) (don't replace '**IBlobProvider**' interface, but replace '**AwsBlobProvider**' class).
- \* '**IAwsBlobNameCalculator**' is used to calculate the full BLOB name (that is explained above). It is implemented by the '**DefaultAwsBlobNameCalculator**' by default.
- \* '**IAmazonS3ClientFactory**' is used create OSS client. It is implemented by the '**DefaultAmazonS3ClientFactory**' by default. You can override/replace it, if you want customize.

#### 7.4.2.7 Create a Custom Provider

##### # BLOB Storing: Creating a Custom Provider

This document explains how you can create a new storage provider for the BLOB storing system with an example.

> Read the [BLOB Storing document](Blob-Storing.md) to understand how to use the BLOB storing system. This document only covers how to create a new storage provider.

##### ## Example Implementation

The first step is to create a class implements the `IBlobProvider` interface or inherit from the `BlobProviderBase` abstract class.

```
```csharp
using System.IO;
using System.Threading.Tasks;
using Volo.Abp.BlobStoring;
using Volo.Abp.DependencyInjection;

namespace AbpDemo
{
    public class MyCustomBlobProvider : BlobProviderBase,
    ITransientDependency
    {
        public override Task SaveAsync(BlobProviderSaveArgs args)
        {
            //TODO...
        }

        public override Task<bool> DeleteAsync(BlobProviderDeleteArgs args)
        {
            //TODO...
        }

        public override Task<bool> ExistsAsync(BlobProviderExistsArgs args)
        {
            //TODO...
        }

        public override Task<Stream> GetOrNullAsync(BlobProviderGetArgs args)
        {
            //TODO...
        }
    }
}...```

```

- * `MyCustomBlobProvider` inherits from the `BlobProviderBase` and overrides the `abstract` methods. The actual implementation is up to you.
- * Implementing `ITransientDependency` registers this class to the [Dependency Injection](Dependency-Injection.md) system as a transient service.

```
> **Notice: Naming conventions are important**. If your class name doesn't end with 'BlobProvider', you must manually register/expose your service for the 'IBlobProvider'.
```

That's all. Now, you can configure containers (inside the 'ConfigureServices' method of your [module](Module-Development-Basics.md)) to use the 'MyCustomBlobProvider' class:

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.ConfigureDefault(container =>
 {
 container.ProviderType = typeof(MyCustomBlobProvider);
 });
}...```

```

> See the [BLOB Storing document](Blob-Storing.md) if you want to configure a specific container.

### ### BlobContainerConfiguration Extension Method

If you want to provide a simpler configuration, create an extension method for the 'BlobContainerConfiguration' class:

```
```csharp
public static class MyBlobContainerConfigurationExtensions
{
    public static BlobContainerConfiguration UseMyCustomBlobProvider(
        this BlobContainerConfiguration containerConfiguration)
    {
        containerConfiguration.ProviderType = typeof(MyCustomBlobProvider);
        return containerConfiguration;
    }
}...```

```

Then you can configure containers easier using the extension method:

```
```csharp
Configure<AbpBlobStoringOptions>(options =>
{
 options.Containers.ConfigureDefault(container =>
 {
 container.UseMyCustomBlobProvider();
 });
}...```

```

### ### Extra Configuration Options

'BlobContainerConfiguration' allows to add/remove provider specific configuration objects. If your provider needs to additional configuration, you can create a wrapper class to the 'BlobContainerConfiguration' for a type-safe configuration option:

```
```csharp

```

```

public class MyCustomBlobProviderConfiguration
{
    public string MyOption1
    {
        get => _containerConfiguration
            .GetConfiguration<string>("MyCustomBlobProvider.MyOption1");
        set => _containerConfiguration
            .SetConfiguration("MyCustomBlobProvider.MyOption1", value);
    }

    private readonly BlobContainerConfiguration _containerConfiguration;

    public MyCustomBlobProviderConfiguration(
        BlobContainerConfiguration containerConfiguration)
    {
        _containerConfiguration = containerConfiguration;
    }
}
...

```

Then you can change the `MyBlobContainerConfigurationExtensions` class like that:

```

```csharp
public static class MyBlobContainerConfigurationExtensions
{
 public static BlobContainerConfiguration UseMyCustomBlobProvider(
 this BlobContainerConfiguration containerConfiguration,
 Action<MyCustomBlobProviderConfiguration> configureAction)
 {
 containerConfiguration.ProviderType = typeof(MyCustomBlobProvider);

 configureAction.Invoke(
 new MyCustomBlobProviderConfiguration(containerConfiguration));
 }

 return containerConfiguration;
}

 public static MyCustomBlobProviderConfiguration
GetMyCustomBlobProviderConfiguration(
 this BlobContainerConfiguration containerConfiguration)
{
 return new MyCustomBlobProviderConfiguration(containerConfiguration);
}
```

```

- * Added an action parameter to the `UseMyCustomBlobProvider` method to allow developers to set the additional options.
- * Added a new `GetMyCustomBlobProviderConfiguration` method to be used inside `MyCustomBlobProvider` class to obtain the configured values.

Then anyone can set the `MyOption1` as shown below:

```

```csharp
Configure<AbpBlobStoringOptions>(options =>
{

```

```

options.Containers.ConfigureDefault(container =>
{
 container.UseMyCustomBlobProvider(provider =>
 {
 provider.MyOption1 = "my value";
 });
});
```

```

Finally, you can access to the extra options using the `GetMyCustomBlobProviderConfiguration` method:

```

```csharp
public class MyCustomBlobProvider : BlobProviderBase, ITransientDependency
{
 public override Task SaveAsync(BlobProviderSaveArgs args)
 {
 var config =
args.Configuration.GetMyCustomBlobProviderConfiguration();
 var value = config.MyOption1;

 //...
 }
}
```

```

Contribute?

If you create a new provider and you think it can be useful for other developers, please consider to [contribute](Contribution/Index.md) to the ABP Framework on GitHub.

7.5 Cancellation Token Provider

Cancellation Token Provider

A `CancellationToken` enables cooperative cancellation between threads, thread pool work items, or `Task` objects. To handle the possible cancellation of the operation, ABP Framework provides `ICancellationTokenProvider` to obtain the `CancellationToken` itself from the source.

> To get more information about `CancellationToken`, see [Microsoft Documentation](<https://docs.microsoft.com/en-us/dotnet/api/system.threading.cancellationtoken>).

ICancellationTokenProvider

`ICancellationTokenProvider` is an abstraction to provide `CancellationToken` for different scenarios.

Generally, you should pass the `CancellationToken` as a parameter for your method to use it. With the `ICancellationTokenProvider` you don't need to pass `CancellationToken` for every method. `ICancellationTokenProvider` can

be injected with the **dependency injection** and provides the token from it's source.

Example:

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Threading;

namespace MyProject
{
 public class MyService : ITransientDependency
 {
 private readonly ICancellationTokenProvider
取消ationTokenProvider;

 public MyService(ICancellationTokenProvider
cancellationTokenProvider)
 {
 _cancellationTokenProvider = cancellationTokenProvider;
 }

 public async Task DoItAsync()
 {
 while (_cancellationTokenProvider.Token.IsCancellationRequested
== false)
 {
 // ...
 }
 }
 }
}..
```

**## Built-in providers**

- `'NullCancellationTokenProvider'`

The `'NullCancellationTokenProvider'` is a built in provider and it supply always `'CancellationToken.None'`.

- `'HttpContextCancellationTokenProvider'`

The `'HttpContextCancellationTokenProvider'` is a built in default provider for ABP Web applications. It simply provides a `'CancellationToken'` that is source of the web request from the `'HttpContext'`.

**## Implementing the ICancellationTokenProvider**

You can easily create your `CancellationTokenProvider` by creating a class that implements the `'ICancellationTokenProvider'` interface, as shown below:

```
```csharp
using System.Threading;

namespace AbpDemo
{
```

```

public class MyCancellationTokenProvider : ICancellationTokenProvider
{
    public CancellationToken Token { get; }

    private MyCancellationTokenProvider()
    {

    }
}

...

```

7.6 CSRF/XSRF & Anti Forgery

CSRF/XSRF & Anti Forgery System

*"*Cross-Site Request Forgery (CSRF) is a type of attack that occurs when a malicious web site, email, blog, instant message, or program causes a user's web browser to perform an unwanted action on a trusted site for which the user is currently authenticated*"*
([OWASP]([https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)_Prevention_Cheat_Sheet](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet))).

ABP Framework completely automates CSRF preventing and works out of the box without any configuration. Read this documentation only if you want to understand it better or need to customize.

The Problem

ASP.NET Core [provides infrastructure](<https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery>) to prevent CSRF attacks by providing a system to **generate** and **validate antiforgery tokens**. However, the standard implementation has a few drawbacks;

Antiforgery token validation is only **enabled for razor pages by default** and not enabled for **HTTP APIs**. You need to enable it yourself for the Controllers. You can use the `'[ValidateAntiForgeryToken]'` attribute for a specific API Controller/Action or the `'[AutoValidateAntiforgeryToken]'` attribute to prevent attacks globally.

Once you enable it;

- * You need to manually add an HTTP header, named `RequestVerificationToken` to every **AJAX request** made in your application. You should care about obtaining the token, saving in the client side and adding to the HTTP header on every HTTP request.
- * All your clients, including **non-browser clients**, should care about obtaining and sending the antiforgery token in every request. In fact, non-browser clients has no CSRF risk and should not care about this.

Especially, the second point is a pain for your clients and unnecessarily consumes your server resources.

> You can read more about the ASP.NET Core antiforgery system in its own [documentation](<https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery>).

The Solution

ABP Framework provides `[AbpValidateAntiForgeryToken]` and `[AbpAutoValidateAntiforgeryToken]` attributes, just like the attributes explained above. `[AbpAutoValidateAntiforgeryToken]` is already added to the global filters, so you should do nothing to enable it for your application.

ABP Framework also automates the following infrastructure;

- * Server side sets a **special cookie**, named `XSRF-TOKEN` by default, that is used make the antiforgery token value available to the browser. This is **done automatically** (by the [application configuration](API/Application-Configuration.md) endpoint). Nothing to do in the client side.
- * In the client side, it reads the token from the cookie and sends it in the **HTTP header** (named `RequestVerificationToken` by default). This is implemented for all the supported UI types.
- * Server side validates the antiforgery token **only for same and cross site requests** made by the browser. It bypasses the validation for non-browser clients.

That's all. The systems works smoothly.

Configuration / Customization

AbpAntiForgeryOptions

`AbpAntiForgeryOptions` is the main [options class](Options.md) to configure the ABP Antiforgery system. It has the following properties;

- * `TokenCookie`: Can be used to configure the cookie details. This cookie is used to store the antiforgery token value in the client side, so clients can read it and sends the value as the HTTP header. Default cookie name is `XSRF-TOKEN`, expiration time is 10 years (yes, ten years! It should be a value longer than the authentication cookie max life time, for the security).
- * `AuthCookieSchemaName`: The name of the authentication cookie used by your application. Default value is `Identity.Application` (which becomes `AspNetCore.Identity.Application` on runtime). The default value properly works with the ABP startup templates. **If you change the authentication cookie name, you also must change this.**
- * `AutoValidate`: The single point to enable/disable the ABP automatic antiforgery validation system. Default value is `true`.
- * `AutoValidateFilter`: A predicate that gets a type and returns a boolean. ABP uses this predicate to check a controller type. If it returns false for a controller type, the controller is excluded from the automatic antiforgery token validation.
- * `AutoValidateIgnoredHttpMethods`: A list of HTTP Methods to ignore on automatic antiforgery validation. Default value: "GET", "HEAD", "TRACE", "OPTIONS". These HTTP Methods are safe to skip antiforgery validation since they don't change the application state.

If you need to change these options, do it in the `ConfigureServices` method of your [module](Module-Development-Basics.md).

Example: Configuring the AbpAntiForgeryOptions

```
```csharp
Configure<AbpAntiForgeryOptions>(options =>
```

```

{
 options.TokenCookie.Expiration = TimeSpan.FromDays(365);
 options.AutoValidateIgnoredHttpMethods.Remove("GET");
 options.AutoValidateFilter =
 type => !type.Namespace.StartsWith("MyProject.MyIgnoredNamespace");
}

```

This configuration;

- \* Sets the antiforgery token expiration time to ~1 year.
- \* Enables antiforgery token validation for GET requests too.
- \* Ignores the controller types in the specified namespace.

### ### AntiforgeryOptions

`AntiforgeryOptions` is the standard [options class](Options.md) of the ASP.NET Core. \*\*You can find all the information about this class in its [own documentation](https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery)\*\*.

'HeaderName' option is especially important for the ABP Framework point of view. Default value of this value is 'RequestVerificationToken' and the clients uses this name while sending the token value in the header. So, if you change this option, you should also arrange your clients to align the change. If you don't have a good reason, leave it as default.

### ### AbpValidateAntiForgeryToken Attribute

If you disable the automatic validation or want to perform the validation for an endpoint that is not validated by default (for example, an endpoint with HTTP GET Method), you can use the `[AbpValidateAntiForgeryToken]` attribute for a \*\*controller type or method\*\* (action).

\*\*Example: Add `[AbpValidateAntiForgeryToken]` to a HTTP GET method\*\*

```

```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.AntiForgeryToken;

namespace MyCompanyName.MyProjectName.Controllers
{
    [Route("api/products")]
    public class ProductController : AbpController
    {
        [HttpGet]
        [AbpValidateAntiForgeryToken]
        public async Task GetAsync()
        {
            //TODO: ...
        }
    }
}

```

```

### ### Angular UI

Angular supports CSRF Token out of box, but the token header name is '`X-XSRF-TOKEN`'. Since ABP Framework follows the ASP.NET Core conventions, it changes this value to '`RequestVerificationToken`' in the core package.

You don't need to make anything unless you need to change the '`AntiforgeryOptions.HeaderName`' as explained before. If you change it, remember to change the header name for the Angular application too. To do that, add an import declaration for the '`HttpClientXsrfModule`' into your root module.

**\*\*Example: Change the header name to `*MyCustomHeaderName`\*\*\***

```
```typescript
@NgModule({
  // ...
  imports: [
    //...
    HttpClientXsrfModule.withOptions({
      cookieName: 'XSRF-TOKEN',
      headerName: 'MyCustomHeaderName'
    })
  ],
})
export class AppModule {}
```

****Note:**** `XSRF-TOKEN` is only valid if both frontend application and APIs run on the same domain. Therefore, when you make a request, you should use a relative path.

For example, let's say your APIs is hosted at '<https://testdomain.com/ws>' and your angular application is hosted at '<https://testdomain.com/admin>'

So if your API request should look like this
`'https://testdomain.com/ws/api/identity/users'`

your '`environment.prod.ts`' has to be as follows:

```
```typescript
export const environment = {
 production: true,
 // ...
 apis: {
 default: {
 url: '/ws', // <- just use the context root here
 // ...
 },
 },
} as Config.Environment;
```

Let's talk about why.

First, take a look at [Angular's code](<https://github.com/angular/angular/blob/master/packages/common/http/src/xsrf.ts#L81>)

It does not intercept any request that starts with `http://` or `https://`. There is a good reason for that. Any cross-site request does not need this token for security. This verification is only valid if the request is made to the same domain from which the web page is served. So, simply put, if you serve everything from a single domain, you just use a relative path.

If you serve your APIs from the root, i.e. no context root (<https://testdomain.com/api/identity/users>), leave `url` empty as follows:

```
```typescript
export const environment = {
  production: true,
  // ...
  apis: {
    default: {
      url: '', // <- should be empty string, not '/'
      // ...
    },
  },
} as Config.Environment;
````
```

## 7.7 Concurrency Check

### ## Concurrency Check

#### ### Introduction

Concurrency Check (also known as **Concurrency Control**) refers to specific mechanisms used to ensure data consistency in the presence of concurrent changes (multiple processes, users access or change the same data in a database at the same time).

There are two commonly used concurrency control mechanisms/approaches:

- \* **Optimistic Concurrency Control**: Optimistic Concurrency Control allows multiple users to attempt to **update** the same record without informing the users that others are also attempting to **update** it.

- \* If a user successfully updates the record, the other users need to get the latest changes for the current record to be able to make changes.

- \* ABP's concurrency check system uses the **Optimistic Concurrency Control**.

- \* **Pessimistic Concurrency Control**: Pessimistic Concurrency Control prevents simultaneous updates to records and uses a locking mechanism. For more information please see [\[here\]](#)(<https://www.martinfowler.com/eaaCatalog/pessimisticOfflineLock.html>).

#### ### Usage

##### #### `IHasConcurrencyStamp` Interface

To enable **concurrency control** to your entity class, you should implement the **`IHasConcurrencyStamp`** interface, directly or indirectly.

```
```csharp
```

```
public interface IHasConcurrencyStamp
{
    public string ConcurrencyStamp { get; set; }
}..
```

* It is the base interface for **concurrency control** and only has a simple property named '`ConcurrencyStamp`'.

* While a new record is **creating**, if the entity implements the '`IHasConcurrencyStamp`' interface, ABP Framework automatically sets a unique value to the **ConcurrencyStamp** property.

* While a record is **updating**, ABP Framework compares the **ConcurrencyStamp** property of the entity with the provided **ConcurrencyStamp** value by the user and if the values match, it automatically updates the **ConcurrencyStamp** property with the new unique value. If there is a mismatch, '`AbpDbConcurrencyException`' is thrown.

****Example: Applying Concurrency Control for the Book Entity****

Implement the '`IHasConcurrencyStamp`' interface for your entity:

```
```csharp
public class Book : Entity<Guid>, IHasConcurrencyStamp
{
 public string ConcurrencyStamp { get; set; }

 //...
}```
```

Also, implement your output and update the DTO classes from the '`IHasConcurrencyStamp`' interface:

```
```csharp
public class BookDto : EntityDto<Guid>, IHasConcurrencyStamp
{
    //...

    public string ConcurrencyStamp { get; set; }
}

public class UpdateBookDto : IHasConcurrencyStamp
{
    //...

    public string ConcurrencyStamp { get; set; }
}```
```

Set the **ConcurrencyStamp** input value to the entity in the **UpdateAsync** method of your application service as below:

```
```csharp
public class BookAppService : ApplicationService, IBookAppService
{
 //...
```

```

 public virtual async Task<BookDto> UpdateAsync(Guid id, UpdateBookDto
input)
{
 var book = await BookRepository.GetAsync(id);

 book.ConcurrencyStamp = input.ConcurrencyStamp;

 //set other input values to the entity ...

 await BookRepository.UpdateAsync(book);
}
```

```

* After that, when multiple users try to update the same record at the same time, the concurrency stamp mismatch occurs and `AbpDbConcurrencyException` is thrown.

Base Classes

[Aggregate Root](./Entities.md#aggregateroot-class) entity classes already implement the `IHasConcurrencyStamp` interface. So, if you are deriving from one of these base classes, you don't need to manually implement the `IHasConcurrencyStamp` interface:

- `AggregateRoot`, `AggregateRoot<TKey>`
- `CreationAuditedAggregateRoot`, `CreationAuditedAggregateRoot<TKey>`
- `AuditedAggregateRoot`, `AuditedAggregateRoot<TKey>`
- `FullAuditedAggregateRoot`, `FullAuditedAggregateRoot<TKey>`

****Example: Applying Concurrency Control for the Book Entity****

You can inherit your entity from one of [the base classes](#base-classes):

```
```csharp
public class Book : FullAuditedAggregateRoot<Guid>
{
 //...
}
```

```

Then, you can implement your output and update the DTO classes from the `IHasConcurrencyStamp` interface:

```
```csharp
public class BookDto : EntityDto<Guid>, IHasConcurrencyStamp
{
 //...

 public string ConcurrencyStamp { get; set; }
}

public class UpdateBookDto : IHasConcurrencyStamp
{
 //...

 public string ConcurrencyStamp { get; set; }
}
```

```

...

Set the **ConcurrencyStamp** input value to the entity in the **UpdateAsync** method of your application service as below:

```
```csharp
public class BookAppService : ApplicationService, IBookAppService
{
 //...

 public virtual async Task<BookDto> UpdateAsync(Guid id, UpdateBookDto
input)
 {
 var book = await BookRepository.GetAsync(id);

 book.ConcurrencyStamp = input.ConcurrencyStamp;

 //set other input values to the entity ...

 await BookRepository.UpdateAsync(book);
 }
}
````
```

After that, when multiple users try to update the same record at the same time, the concurrency stamp mismatch occurs and `'AbpDbConcurrencyException'` is thrown. You can either handle the exception manually or let the ABP Framework handle it for you.

ABP Framework shows a user-friendly error message as in the image below, if you don't handle the exception manually.

![Optimistic Concurrency](./images/optimistic-concurrency.png)

7.8 Current User

Current User

It is very common to retrieve the information about the logged in user in a web application. The current user is the active user related to the current request in a web application.

ICurrentUser

`'ICurrentUser'` is the main service to get info about the current active user.

Example: [\[Injecting\]](#)(Dependency-Injection.md) the `'ICurrentUser'` into a service:

```
```csharp
using System;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Users;

namespace AbpDemo
{
```

```

public class MyService : ITransientDependency
{
 private readonly ICurrentUser _currentUser;

 public MyService(ICurrentUser currentUser)
 {
 _currentUser = currentUser;
 }

 public void Foo()
 {
 Guid? userId = _currentUser.Id;
 }
}
...

```

Common base classes have already injected this service as a base property. For example, you can directly use the `CurrentUser` property in an [application service](Application-Services.md):

```

```csharp
using System;
using Volo.Abp.Application.Services;

namespace AbpDemo
{
    public class MyAppService : ApplicationService
    {
        public void Foo()
        {
            Guid? userId = CurrentUser.Id;
        }
    }
}
...

```

Properties

Here are the fundamental properties of the `ICurrentUser` interface:

- * **IsAuthenticated** (bool): Returns `true` if the current user has logged in (authenticated). If the user has not logged in then `Id` and `UserName` returns `null`.
- * **Id** (Guid?): Id of the current user. Returns `null`, if the current user has not logged in.
- * **UserName** (string): User name of the current user. Returns `null`, if the current user has not logged in.
- * **TenantId** (Guid?): Tenant Id of the current user, which can be useful for a [multi-tenant](Multi-Tenancy.md) application. Returns `null`, if the current user is not assigned to a tenant.
- * **Email** (string): Email address of the current user. Returns `null`, if the current user has not logged in or not set an email address.
- * **EmailVerified** (bool): Returns `true`, if the email address of the current user has been verified.
- * **PhoneNumber** (string): Phone number of the current user. Returns `null`, if the current user has not logged in or not set a phone number.

```
* **PhoneNumberVerified** (bool): Returns 'true', if the phone number of the current user has been verified.  
* **Roles** (string[]): Roles of the current user. Returns a string array of the role names of the current user.
```

Methods

'ICurrentUser' is implemented on the 'ICurrentPrincipalAccessor' (see the section below) and works with the claims. So, all of the above properties are actually retrieved from the claims of the current authenticated user.

'ICurrentUser' has some methods to directly work with the claims, if you have custom claims or get other non-common claim types.

- * **FindClaim**: Gets a claim with the given name. Returns 'null' if not found.
- * **FindClaims**: Gets all the claims with the given name (it is allowed to have multiple claim values with the same name).
- * **GetAllClaims**: Gets all the claims.
- * **IsInRole**: A shortcut method to check if the current user is in the specified role.

Beside these standard methods, there are some extension methods:

- * **FindClaimValue**: Gets the value of the claim with the given name, or 'null' if not found. It has a generic overload that also casts the value to a specific type.
- * **GetId**: Returns 'Id' of the current user. If the current user has not logged in, it throws an exception (instead of returning 'null') . Use this only if you are sure that the user has already authenticated in your code context.

Authentication & Authorization

'ICurrentUser' works independently of how the user is authenticated or authorized. It seamlessly works with any authentication system that works with the current principal (see the section below).

ICurrentPrincipalAccessor

'ICurrentPrincipalAccessor' is the service that should be used (by the ABP Framework and your application code) whenever the current principal of the current user is needed.

For a web application, it gets the 'User' property of the current 'HttpContext'. For a non-web application, it returns the 'Thread.CurrentPrincipal'.

> You generally don't need to use this low level 'ICurrentPrincipalAccessor' service and just directly work with the 'ICurrentUser' explained above.

Basic Usage

You can inject 'ICurrentPrincipalAccessor' and use the 'Principal' property to the the current principal:

```
```csharp  
public class MyService : ITransientDependency
```

```

{
 private readonly ICurrentPrincipalAccessor _currentPrincipalAccessor;

 public MyService(ICurrentPrincipalAccessor currentPrincipalAccessor)
 {
 _currentPrincipalAccessor = currentPrincipalAccessor;
 }

 public void Foo()
 {
 var allClaims = _currentPrincipalAccessor.Principal.Claims.ToList();
 //...
 }
}
...

```

### ### Changing the Current Principal

Current principal is not something you want to set or change, except at some advanced scenarios. If you need it, use the `Change` method of the `ICurrentPrincipalAccessor`. It takes a `ClaimsPrincipal` object and makes it "current" for a scope.

Example:

```

```csharp
public class MyAppService : ApplicationService
{
    private readonly ICurrentPrincipalAccessor _currentPrincipalAccessor;

    public MyAppService(ICurrentPrincipalAccessor currentPrincipalAccessor)
    {
        _currentPrincipalAccessor = currentPrincipalAccessor;
    }

    public void Foo()
    {
        var newPrincipal = new ClaimsPrincipal(
            new ClaimsIdentity(
                new Claim[]
                {
                    new Claim(AbpClaimTypes.UserId,
Guid.NewGuid().ToString(),
                    new Claim(AbpClaimTypes.UserName, "john"),
                    new Claim("MyCustomClaim", "42")
                }
            )
        );
        using (_currentPrincipalAccessor.Change(newPrincipal))
        {
            var userName = CurrentUser.UserName; //returns "john"
            //...
        }
    }
}
...

```

Use the `Change` method always in a `using` statement, so it will be restored to the original value after the `using` scope ends.

This can be a way to simulate a user login for a scope of the application code, however try to use it carefully.

AbpClaimTypes

`AbpClaimTypes` is a static class that defines the names of the standard claims and used by the ABP Framework.

* Default values for the `UserName`, `UserId`, `Role` and `Email` properties are set from the

[[System.Security.Claims.ClaimTypes](https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claimtypes)](https://docs.microsoft.com/en-us/dotnet/api/system.security.claims.claimtypes) class, but you can change them.

* Other properties, like `EmailVerified`, `PhoneNumber`, `TenantId`... are defined by the ABP Framework by following the standard names wherever possible.

It is suggested to use properties of this class instead of magic strings for claim names.

7.9 Data Filtering

Data Filtering

[[Volo.Abp.Data](https://www.nuget.org/packages/Volo.Abp.Data)](https://www.nuget.org/packages/Volo.Abp.Data) package defines services to automatically filter data on querying from a database.

Pre-Defined Filters

ABP defines some filters out of the box.

ISoftDelete

Used to mark an [[entity](#)](Entities.md) as deleted instead of actually deleting it. Implement the `ISoftDelete` interface to make your entity "soft delete".

Example:

```
```csharp
using System;
using Volo.Abp;
using Volo.Abp.Domain.Entities;

namespace Acme.BookStore
{
 public class Book : AggregateRoot<Guid>, ISoftDelete
 {
 public string Name { get; set; }

 public bool IsDeleted { get; set; } //Defined by ISoftDelete
 }
}
```

....

`**ISoftDelete**` defines the `IsDeleted` property. When you delete a book using [repositories](Repositories.md), ABP automatically sets `IsDeleted` to true and protects it from actual deletion (you can also manually set the `IsDeleted` property to true if you need). In addition, it \*\*automatically filters deleted entities\*\* when you query the database.

> `ISoftDelete` filter is enabled by default and you can not get deleted entities from database unless you explicitly disable it. See the `IDataFilter` service below.

> Soft-delete entities can be hard-deleted when you use `HardDeleteAsync` method on the repositories.

### ### IMultiTenant

[Multi-tenancy](Multi-Tenancy.md) is an efficient way of creating SaaS applications. Once you create a multi-tenant application, you typically want to isolate data between tenants. Implement `IMultiTenant` interface to make your entity "multi-tenant aware".

Example:

```
```csharp
using System;
using Volo.Abp;
using Volo.Abp.Domain.Entities;
using Volo.Abp.MultiTenancy;

namespace Acme.BookStore
{
    public class Book : AggregateRoot<Guid>, ISoftDelete, IMultiTenant
    {
        public string Name { get; set; }

        public bool IsDeleted { get; set; } //Defined by ISoftDelete

        public Guid? TenantId { get; set; } //Defined by IMultiTenant
    }
}...```

```

`IMultiTenant` interface defines the `TenantId` property which is then used to automatically filter the entities for the current tenant. See the [Multi-tenancy](Multi-Tenancy.md) document for more.

IDataFilter Service: Enable/Disable Data Filters

You can control the filters using `IDataFilter` service.

Example:

```
```csharp
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp;```

```

```

using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore
{
 public class MyBookService : ITransientDependency
 {
 private readonly IDataFilter _dataFilter;
 private readonly IRepository<Book, Guid> _bookRepository;

 public MyBookService(
 IDataFilter dataFilter,
 IRepository<Book, Guid> bookRepository)
 {
 _dataFilter = dataFilter;
 _bookRepository = bookRepository;
 }

 public async Task<List<Book>> GetAllBooksIncludingDeletedAsync()
 {
 //Temporary disable the ISoftDelete filter
 using (_dataFilter.Disable<ISoftDelete>())
 {
 return await _bookRepository.GetListAsync();
 }
 }
 }
}
```


* [Inject](Dependency-Injection.md) the `IDataFilter` service to your class.  

* Use the `Disable` method within a `using` statement to create a code block where the `ISoftDelete` filter is disabled inside it.


```

In addition to the `Disable<T>()` method;

- * `IDataFilter.Enable<T>()` method can be used to enable a filter. `Enable` and `Disable` methods can be used in a **nested** way to define inner scopes.
- * `IDataFilter.IsEnabled<T>()` can be used to check whether a filter is currently enabled or not.

> Always use the `Disable` and `Enable` methods it inside a `using` block to guarantee that the filter is reset to its previous state.

The Generic IDataFilter Service

`IDataFilter` service has a generic version, `IDataFilter<TFilter>` that injects a more restricted and explicit data filter based on the filter type.

```
```csharp
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
```

```

using Volo.Abp.Domain.Repositories;

namespace Acme.BookStore
{
 public class MyBookService : ITransientDependency
 {
 private readonly IDataFilter<ISoftDelete> _softDeleteFilter;
 private readonly IRepository<Book, Guid> _bookRepository;

 public MyBookService(
 IDataFilter<ISoftDelete> softDeleteFilter,
 IRepository<Book, Guid> bookRepository)
 {
 _softDeleteFilter = softDeleteFilter;
 _bookRepository = bookRepository;
 }

 public async Task<List<Book>> GetAllBooksIncludingDeletedAsync()
 {
 //Temporary disable the ISoftDelete filter
 using (_softDeleteFilter.Disable())
 {
 return await _bookRepository.GetListAsync();
 }
 }
 }
}

* This usage determines the filter type while injecting the `IDataFilter<T>` service.
* In this case you can use the `Disable()` and `Enable()` methods without specifying the filter type.

```

#### ## AbpDataFilterOptions

`AbpDataFilterOptions` can be used to [set options](Options.md) for the data filter system.

The example code below disables the `ISoftDelete` filter by default which will cause to include deleted entities when you query the database unless you explicitly enable the filter:

```
```csharp
Configure<AbpDataFilterOptions>(options =>
{
    options.DefaultStates[typeof(ISoftDelete)] = new
DataFilterState(isEnabled: false);
});
```

> Carefully change defaults for global filters, especially if you are using a pre-built module which might be developed assuming the soft delete filter is turned on by default. But you can do it for your own defined filters safely.

Defining Custom Filters

Defining and implementing a new filter highly depends on the database provider. ABP implements all pre-defined filters for all database providers.

When you need it, start by defining an interface (like `ISoftDelete` and `IMultiTenant`) for your filter and implement it for your entities.

Example:

```
```csharp
public interface IIIsActive
{
 bool IsActive { get; }
}...```

```

Such an `IIIsActive` interface can be used to filter active/passive data and can be easily implemented by any [entity](Entities.md):

```
```csharp
public class Book : AggregateRoot<Guid>, IIIsActive
{
    public string Name { get; set; }

    public bool IsActive { get; set; } //Defined by IIIsActive
}...```

```

EntityFramework Core

ABP uses [EF Core's Global Query Filters](<https://docs.microsoft.com/en-us/ef/core/querying/filters>) system for the [EF Core Integration](Entity-Framework-Core.md). So, it is well integrated to EF Core and works as expected even if you directly work with `DbContext`.

Best way to implement a custom filter is to override `ShouldFilterEntity` and `CreateFilterExpression` method for your `DbContext`. Example:

```
```csharp
protected bool IsActiveFilterEnabled => DataFilter?.IsEnabled<IIIsActive>() ??
false;

protected override bool ShouldFilterEntity< TEntity >(IMutableEntityType
entityType)
{
 if (typeof(IIIsActive).IsAssignableFrom(typeof(TEntity)))
 {
 return true;
 }

 return base.ShouldFilterEntity< TEntity >(entityType);
}

protected override Expression< Func< TEntity, bool >>
CreateFilterExpression< TEntity >()
{
 var expression = base.CreateFilterExpression< TEntity >();

 if (typeof(IIIsActive).IsAssignableFrom(typeof(TEntity)))

```

```

 {
 Expression<Func< TEntity, bool>> isActiveFilter =
 e => !IsActiveFilterEnabled || EF.Property<bool>(e, "IsActive");
 expression = expression == null
 ? isActiveFilter
 : CombineExpressions(expression, isActiveFilter);
 }

 return expression;
}
...

```

\* Added a `IsActiveFilterEnabled` property to check if `IIsActive` is enabled or not. It internally uses the `IDataFilter` service introduced before.  
\* Overrided the `ShouldFilterEntity` and `CreateFilterExpression` methods, checked if given entity implements the `IIsActive` interface and combines the expressions if necessary.

In addition you can also use `HasAbpQueryFilter` to set a filter for an entity. It will combine your filter with ABP EF Core builtin global query filters.

```

```csharp
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);

    modelBuilder.Entity<MyEntity>(b =>
    {
        b.HasAbpQueryFilter(e => e.Name.StartsWith("abp"));
    });
}
...

```

MongoDB

ABP abstracts the `IMongoRepositoryFilterer` interface to implement data filtering for the [MongoDB Integration](MongoDB.md), it works only if you use the repositories properly. Otherwise, you should manually filter the data.

Currently, the best way to implement a data filter for the MongoDB integration is to create a derived class of `MongoDbRepositoryFilterer` and override `FilterQueryable`. Example:

```

```csharp
[ExposeServices(typeof(IMongoRepositoryFilterer<Book, Guid>))]
public class BookMongoDbRepositoryFilterer : MongoDbRepositoryFilterer<Book,
Guid>, ITransientDependency
{
 public BookMongoDbRepositoryFilterer(
 IDataFilter dataFilter,
 ICurrentTenant currentTenant) :
 base(dataFilter, currentTenant)
 {

 }

 public override IQueryable FilterQueryable<TQueryable>(TQueryable query)
 {

```

```

 if (DataFilter.IsEnabled<IIIsActive>())
 {
 return (TQueryable)query.Where(x => x.IsActive);
 }

 return base.FilterQueryable(query);
}
}...

```

This example implements it only for the `Book` entity. If you want to implement for all entities (those implement the `IIIsActive` interface), create your own custom MongoDB repository filterer base class and override the `AddGlobalFilters` as shown below:

```

```csharp
public abstract class MyMongoRepository<TMongoDbContext, TEntity, TKey> : 
MongoDbRepository<TMongoDbContext, TEntity, TKey>
    where TMongoDbContext : IAbpMongoDbContext
    where TEntity : class, IEntity<TKey>
{
    protected MyMongoRepository(IMongoDbContextProvider<TMongoDbContext>
dbContextProvider)
        : base(dbContextProvider)
    {

    }

    protected override void AddGlobalFilters(List<FilterDefinition<

```

7.10 Data Seeding

```
# Data Seeding
```

Introduction

Some applications (or modules) using a database may need to have some **initial data** to be able to properly start and run. For example, an **admin user** & roles must be available at the beginning. Otherwise you can not **login** to the application to create new users and roles.

Data seeding is also useful for [testing](Testing.md) purpose, so your automatic tests can assume some initial data available in the database.

Why a Data Seed System?

While EF Core Data Seeding system provides a way, it is very limited and doesn't cover production scenarios. Also, it is only for EF Core.

ABP Framework provides a data seed system that is;

- * **Modular**: Any [module](Module-Development-Basics.md) can silently contribute to the data seeding process without knowing and effecting each other. In this way, a module seeds its own initial data.
- * **Database Independent**: It is not only for EF Core, it also works for other database providers (like [MongoDB](MongoDB.md)).
- * **Production Ready**: It solves the problems on production environments. See the "**On Production**" section below.
- * **Dependency Injection**: It takes the full advantage of dependency injection, so you can use any internal or external service while seeding the initial data. Actually, you can do much more than data seeding.

IDataSeedContributor

'**IDataSeedContributor**' is the interface that should be implemented in order to seed data to the database.

Example: Seed one initial book to the database if there is no book**

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Guids;

namespace Acme.BookStore
{
 public class BookStoreDataSeedContributor
 : IDataSeedContributor, ITransientDependency
 {
 private readonly IRepository<Book, Guid> _bookRepository;
 private readonly IGuidGenerator _guidGenerator;
 private readonly ICurrentTenant _currentTenant;

 public BookStoreDataSeedContributor(
 IRepository<Book, Guid> bookRepository,
 IGuidGenerator guidGenerator,
 ICurrentTenant currentTenant)
```

```

 {
 _bookRepository = bookRepository;
 _guidGenerator = guidGenerator;
 _currentTenant = currentTenant;
 }

 public async Task SeedAsync(DataSeedContext context)
 {
 using (_currentTenant.Change(context?.TenantId))
 {
 if (await _bookRepository.GetCountAsync() > 0)
 {
 return;
 }

 var book = new Book(
 id: _guidGenerator.Create(),
 name: "The Hitchhiker's Guide to the Galaxy",
 type: BookType.ScienceFiction,
 publishDate: new DateTime(1979, 10, 12),
 price: 42
);
 await _bookRepository.InsertAsync(book);
 }
 }
}
```


* `IDataSeedContributor` defines the `SeedAsync` method to execute the **data seed logic**.  

* It is typical to **check database** if the seeding data is already present.  

* You can **inject** service and perform any logic needed to seed the data.


```

> Data seed contributors are automatically discovered by the ABP Framework and executed as a part of the data seed process.

DataSeedContext

`DataSeedContext` contains `TenantId` if your application is [multi-tenant](Multi-Tenancy.md), so you can use this value while inserting data or performing custom logic based on the tenant.

`DataSeedContext` also contains name-value style configuration parameters for passing to the seeder contributors from the `IDataSeeder`.

Modularity

An application can have multiple data seed contributor (`IDataSeedContributor`) class. So, any reusable module can also implement this interface to seed its own initial data.

For example, the [Identity Module](Modules/Identity.md) has a data seed contributor that creates an admin role and admin user and assign all the permissions.

IDataSeeder

> You typically never need to directly use the '`IDataSeeder`' service since it is already done if you've started with the [application startup template](Startup-Templates/Application.md). But its suggested to read it to understand the design behind the data seed system.

`'IDataSeeder'` is the main service that is used to seed initial data. It is pretty easy to use;

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IDataSeeder _dataSeeder;

 public MyService(IDataSeeder dataSeeder)
 {
 _dataSeeder = dataSeeder;
 }

 public async Task FooAsync()
 {
 await _dataSeeder.SeedAsync();
 }
}
```

```

You can [inject](Dependency-Injection.md) the '`IDataSeeder`' and use it to seed the initial data when you need. It internally calls all the '`IDataSeedContributor`' implementations to complete the data seeding.

It is possible to send named configuration parameters to the '`SeedAsync`' method as shown below:

```
```csharp
await _dataSeeder.SeedAsync(
 new DataSeedContext()
 .WithProperty("MyProperty1", "MyValue1")
 .WithProperty("MyProperty2", 42)
);
```

```

Then the data seed contributors can access to these properties via the '`DataSeedContext`' explained before.

If a module needs to a parameter, it should be declared on the [module documentation](Modules/Index.md). For example, the [Identity Module](Modules/Identity.md) can use '`AdminEmail`' and '`AdminPassword`' parameters if you provide (otherwise uses the default values).

Separate Unit Of Works

The default seed will be in a unit of work and may use transactions. If there are multiple '`IDataSeedContributor`' or too much data written, it may cause a database timeout error.

We provide an extension method of '`SeedInSeparateUowAsync`' for the '`IDataSeeder`' service to create a separate unit of work for each '`IDataSeedContributor`'.

```
```csharp
public static Task SeedInSeparateUowAsync(this IDataSeeder seeder, Guid?
tenantId = null, AbpUnitOfWorkOptions options = null, bool requiresNew =
false)
```

```

Where & How to Seed Data?

It is important to understand where & how to execute the `'IDataSeeder.SeedAsync()'`?

On Production

The [\[application startup template\]](#)(Startup-Templates/Application.md) comes with a `*YourProjectName**.DbMigrator**` project (Acme.BookStore.DbMigrator on the picture below), which is a `**console application**` that is responsible to `**migrate**` the database schema (for relational databases) and `**seed**` the initial data:

[!\[bookstore-visual-studio-solution-v3\]\(images/bookstore-visual-studio-solution-v3.png\)](#)

This console application is properly configured for you. It even supports `**multi-tenant**` scenarios where each tenant has its own database (`migrates & seeds` all necessary databases).

It is expected to run this DbMigrator application whenever you `**deploy a new version**` of your solution to the server. It will `migrate` your `**database schema**` (create new tables/fields... etc.) and `**seed new initial data**` needed to properly run the new version of your solution. Then you can `deploy/start` your actual application.

Even if you are using MongoDB or another NoSQL database (that doesn't need to `schema migrations`), it is recommended to use the DbMigrator application to `seed` your data or perform your data migration.

Having such a separate console application has several advantages;

- * You can `**run it before**` updating your application, so your application will run on the ready database.
- * Your application `**starts faster**` compared to if it seeds the initial data itself.
- * Your application can properly run on a `**clustered environment**` (where multiple instances of your application run concurrently). If you seed data on application startup you would have conflicts in this case.

On Development

We suggest the same way on development. Run the DbMigrator console application whenever you [\[create a database migration\]](#)(<https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/migrations/>) (using EF Core `'Add-Migration'` command, for example) or change the data seed code (will be explained later).

> You can continue to use the standard `'Update-Database'` command for EF Core, but it will not seed if you've created a new seed data.

On Testing

You probably want to seed the data also for automated [testing](Testing.md), so want to use the `IDataSeeder.SeedAsync()`. In the [application startup template](Startup-Templates/Application.md), it is done in the [OnApplicationInitialization](Module-Development-Basics.md) method of the *YourProjectName*TestBaseModule class of the TestBase project.

In addition to the standard seed data (that is also used on production), you may want to seed additional data unique to the automated tests. If so, you can create a new data seed contributor in the test project to have more data to work on.

7.11 Distributed Locking

7.12 Email Sending

7.12.1 Email Sending System

Email Sending

ABP Framework provides various services, settings and integrations for sending emails;

- * Provides `IEmailSender` service that is used to send emails.
- * Defines [settings](Settings.md) to configure email sending.
- * Integrates to the [background job system](Background-Jobs.md) to send emails via background jobs.
- * Provides [MailKit integration](MailKit.md) package.

Installation

> This package is already installed if you are using the [application startup template](Startup-Templates/Application.md).

It is suggested to use the [ABP CLI](CLI.md) to install this package. Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Emailing
```
```

If you haven't done it yet, you first need to install the ABP CLI. For other installation options, see [the package description page](<https://abp.io/package-detail/Volo.Abp.Emailing>).

Sending Emails

IEmailSender

[Inject](Dependency-Injection.md) the `IEmailSender` into any service and use the `SendAsync` method to send emails.

Example

```

```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Emailing;

namespace MyProject
{
 public class MyService : ITransientDependency
 {
 private readonly IEmailSender _emailSender;

 public MyService(IEmailSender emailSender)
 {
 _emailSender = emailSender;
 }

 public async Task DoItAsync()
 {
 await _emailSender.SendAsync(
 "target@domain.com", // target email address
 "Email subject", // subject
 "This is email body..." // email body
);
 }
 }
}
```

```

`'SendAsync'` method has overloads to supply more parameters like;

- * **from**: You can set this as the first argument to set a sender email address. If not provided, the default sender address is used (see the email settings below).

- * **isBodyHtml**: Indicates whether the email body may contain HTML tags.
Default: true.

> `IEmailSender` is the suggested way to send emails, since it makes your code provider independent.

MailMessage

In addition to primitive parameters, you can pass a **standard `'MailMessage'` object** ([[see](#)](<https://docs.microsoft.com/en-us/dotnet/api/system.net.mail.mailmessage>)) to the `'SendAsync'` method to set more options, like adding attachments.

ISmtpEmailSender

Sending emails is implemented by the standard `'SmtpClient'` class ([[see](#)](<https://docs.microsoft.com/en-us/dotnet/api/system.net.smtpclient>)) by default. The implementation class is the `'SmtpEmailSender'`. This class also expose the `'ISmtpEmailSender'` service (in addition to the `'IEmailSender'`).

Most of the time you want to directly use the `'IEmailSender'` to make your code provider independent. However, if you want to create an `'SmtpClient'` object with the same email settings, you can inject the `'ISmtpEmailSender'`

and use its `'BuildClientAsync'` method to obtain a `'SmtpClient'` object and send the email yourself.

Queueing Emails / Background Jobs

`'IEmailSender'` has a `'QueueAsync'` method that can be used to add emails to the background job queue to send them in a background thread. In this way, you don't take time of the user by waiting to send the email. `'QueueAsync'` method gets the same arguments with the `'SendAsync'` method.

Queueing emails tolerates errors since the background job system has re-try mechanism to overcome temporary network/server problems.

See the [\[background jobs document\]](#)(Background-Jobs.md) for more about the background job system.

Email Settings

Email sending uses the [\[setting system\]](#)(Settings.md) to define settings and get the values of these settings on the runtime.

`'Volo.Abp.Emailing.EmailSettingNames'` defines constants for the setting names, just listed below:

- * ****Abp.Mailing.DefaultFromAddress****: Used as the sender's email address when you don't specify a sender when sending emails (just like in the example above).
- * ****Abp.Mailing.DefaultFromDisplayName****: Used as the sender's display name when you don't specify a sender when sending emails (just like in the example above).
- * ****Abp.Mailing.Smtp.Host****: The IP/Domain of the SMTP server (default: 127.0.0.1).
- * ****Abp.Mailing.Smtp.Port****: The Port of the SMTP server (default: 25).
- * ****Abp.Mailing.Smtp.UserName****: Username, if the SMTP server requires authentication.
- * ****Abp.Mailing.Smtp.Password****: Password, if the SMTP server requires authentication. ****This value is encrypted ****(see the section below).
- * ****Abp.Mailing.Smtp.Domain****: Domain for the username, if the SMTP server requires authentication.
- * ****Abp.Mailing.Smtp.EnableSsl****: A value that indicates if the SMTP server uses SSL or not ("true" or "false". Default: "false").
- * ****Abp.Mailing.Smtp.UseDefaultCredentials****: If true, uses default credentials instead of the provided username and password ("true" or "false". Default: "true").

Email settings can be managed from the ***Settings Page*** of the [\[Setting Management\]](#)(Modules/Setting-Management.md) module:

![email-settings](images/email-settings.png)

> Setting Management module is already installed if you've created your solution from the ABP Startup template.

If you don't use the Setting Management module, you can simply define the settings inside your `'appsettings.json'` file:

```
```json
"Settings": {
 "Abp.Mailing.Smtp.Host": "127.0.0.1",
```

```

 "Abp.Mailing.Smtp.Port": "25",
 "Abp.Mailing.Smtp.UserName": "",
 "Abp.Mailing.Smtp.Password": "",
 "Abp.Mailing.Smtp.Domain": "",
 "Abp.Mailing.Smtp.EnableSsl": "false",
 "Abp.Mailing.Smtp.UseDefaultCredentials": "true",
 "Abp.Mailing.DefaultFromAddress": "noreply@abp.io",
 "Abp.Mailing.DefaultFromDisplayName": "ABP application"
}
```

```

You can set/change these settings programmatically using the `'ISettingManager'` and store values in a database. See the [[setting system document](#)](Settings.md) to understand the setting system better.

[### Encrypt the SMTP Password](#)

`Abp.Mailing.Smtp.Password` must be an **encrypted** value. If you use the `'ISettingManager'` to set the password, you don't have to worry. It internally encrypts the values on set and decrypts on get.

If you use the `'appsettings.json'` to store the password, you should manually inject the `'ISettingEncryptionService'` and use its `'Encrypt'` method to obtain an encrypted value. This can be done by creating a simple code in your application. Then you can delete the code. As better, you can create a UI in your application to configure the email settings. In this case, you can directly use the `'ISettingManager'` without worrying the encryption.

[### ISsmtpEmailSenderConfiguration](#)

If you don't want to use the setting system to store the email sending configuration, you can replace the `'ISsmtpEmailSenderConfiguration'` service with your own implementation to get the configuration from any other source. `'ISsmtpEmailSenderConfiguration'` is implemented by the `'SmtpEmailSenderConfiguration'` by default, which gets the configuration from the setting system as explained above.

[## Text Template Integration](#)

ABP Framework provides a strong and flexible [[text templating system](#)](Text-Templating.md). You can use the text templating system to create dynamic email contents. Inject the `'ITemplateRenderer'` and use the `'RenderAsync'` to render a template. Then use the result as the email body.

While you can define and use your own text templates, email sending system provides two simple built-in text templates.

****Example: Use the standard and simple message template to send emails****

```

```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Emailing;
using Volo.Abp.Emailing.Templates;
using Volo.Abp.TextTemplating;

namespace Acme.BookStore.Web
{

```

```

public class MyService : ITransientDependency
{
 private readonly IEmailSender _emailSender;
 private readonly ITemplateRenderer _templateRenderer;

 public MyService(
 IEmailSender emailSender,
 ITemplateRenderer templateRenderer)
 {
 _emailSender = emailSender;
 _templateRenderer = templateRenderer;
 }

 public async Task DoItAsync()
 {
 var body = await _templateRenderer.RenderAsync(
 StandardEmailTemplates.Message,
 new
 {
 message = "This is email body..."
 }
);

 await _emailSender.SendAsync(
 "target-address@domain.com",
 "Email subject",
 body
);
 }
}
...

```

The resulting email body will be shown below:

```

```html
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
</head>
<body>
    This is email body...
</body>
</html>
```

```

Emailing system defines the built-in text templates with the given names:

"\*\*Abp.StandardEmailTemplates.Message\*\*" is simplest template that has a text message:

```

```html
{{model.message}}
```

```

This template uses the "Abp.StandardEmailTemplates.Layout" as its layout.

**```Abp.StandardEmailTemplates.Layout```** is a simple template to provide an HTML document layout:

```
```html
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
</head>
<body>
    {{{{content}}}}
</body>
</html>
````
```

The final rendered message was shown above.

> These template names are constants defined in the '[`Volo.Abp.Emailing.Templates.StandardEmailTemplates`](#)' class.

### [### Overriding/Replacing the Standard Templates](#)

You typically want to replace the standard templates with your own ones, so you can prepare a branded email messages. To do that, you can use the power of the [\[virtual file system\]\(Virtual-File-System.md\)](#) (VFS) or replace them in your own template definition provider.

Paths of the templates in the virtual file system are shown below:

- \* `'/Volo/Abp/Emailing/Templates/Layout.tpl'`
- \* `'/Volo/Abp/Emailing/Templates/Message.tpl'`

If you add files to the same location in the virtual file system, your files will override them.

Templates are inline localized, that means you can take the power of the [\[localization system\]\(Localization.md\)](#) to make your templates multi-cultural.

See the [\[text templating system\]\(Text-Templating.md\)](#) document for details.

> Notice that you can define and use your own templates for your application, rather than using the standard simple templates. These standard templates are mostly for reusable modules where they don't define their own templates but rely on the built-in ones. This makes easy to customize emails sent by the used modules, by just overriding the standard email layout template.

### [## NullEmailSender](#)

[`NullEmailSender`](#) is a built-in class that implements the [`IEmailSender`](#), but writes email contents to the [\[standard log system\]\(Logging.md\)](#), rather than actually sending the emails.

This class can be useful especially in development time where you generally don't want to send real emails. The [\[application startup template\]\(Startup-Templates/Application.md\)](#) already uses this class in the **\*\*DEBUG mode\*\*** with the following configuration in the domain layer:

```
```csharp
```

```
#if DEBUG
    context.Services.Replace(ServiceDescriptor.Singleton<IEmailSender,
    NullEmailSender>());
#endif
```
```

So, don't confuse if you don't receive emails on DEBUG mode. Emails will be sent as expected on production (RELEASE mode). Remove these lines if you want to send real emails on DEBUG too.

#### ## See Also

- \* [\[MailKit integration for sending emails\]\(MailKit.md\)](#)

### 7.12.2 MailKit Integration

#### # MailKit Integration

[\[MailKit\]](#)(<http://www.mimekit.net/>) is a cross-platform, popular open source mail client library for .net. ABP Framework provides an integration package to use the MailKit as the [\[email sender\]\(Emailing.md\)](#).

#### ## Installation

It is suggested to use the [\[ABP CLI\]\(CLI.md\)](#) to install this package. Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.MailKit
```
```

If you haven't done it yet, you first need to install the ABP CLI. For other installation options, see [\[the package description page\]\(https://abp.io/package-detail/Volo.Abp.MailKit\)](#).

#### ## Sending Emails

##### ### IEmailSender

[\[Inject\]\(Dependency-Injection.md\)](#) the standard `IEmailSender` into any service and use the `SendAsync` method to send emails. See the [\[email sending document\]\(Emailing.md\)](#) for details.

> `IEmailSender` is the suggested way to send emails even if you use MailKit, since it makes your code provider independent.

##### ### IMailKitSmtpEmailSender

MailKit package also exposes the `IMailKitSmtpEmailSender` service that extends the `IEmailSender` by adding the `BuildClientAsync()` method. This method can be used to obtain a `MailKit.Net.Smtp.SmtpClient` object that can be used to perform MailKit specific operations.

#### ## Configuration

MailKit integration package uses the same settings defined by the email sending system. So, refer to the [[email sending document](#)](Emailing.md) for the settings.

In addition to the standard settings, this package defines `AbpMailKitOptions` as a simple [[options](#)](Options.md) class. This class defines only one options:

\* **SecureSocketOption**: Used to set one of the `SecureSocketOptions`. Default: `null` (uses the defaults).

**Example: Use *SecureSocketOptions.SslOnConnect***

```
```csharp
Configure<AbpMailKitOptions>(options =>
{
    options.SecureSocketOption = SecureSocketOptions.SslOnConnect;
});
```

Refer to the [[MailKit documentation](#)](http://www.mimekit.net/) to learn more about this option.

See Also

* [[Email sending](#)](Emailing.md)

7.13 Event Bus

7.13.1 Overall

Event Bus

An event bus is a mediator that transfers a message from a sender to a receiver. In this way, it provides a loosely coupled communication way between objects, services and applications.

Event Bus Types

ABP Framework provides two type of event buses;

* **[Local Event Bus]**(Local-Event-Bus.md) is suitable for in-process messaging.

* **[Distributed Event Bus]**(Distributed-Event-Bus.md) is suitable for inter-process messaging, like microservices publishing and subscribing to distributed events.

7.13.2 Local Event Bus

Local Event Bus

The Local Event Bus allows services to publish and subscribe to **in-process events**. That means it is suitable if two services (publisher and subscriber) are running in the same process.

Publishing Events

There are two ways of publishing local events explained in the following sections.

Publishing Events Using the `ILocalEventBus`

`'ILocalEventBus'` can be [injected](Dependency-Injection.md) and used to publish a local event.

Example: Publish a local event when the stock count of a product changes

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.EventBus.Local;

namespace AbpDemo
{
 public class MyService : ITransientDependency
 {
 private readonly ILocalEventBus _localEventBus;

 public MyService(ILocalEventBus localEventBus)
 {
 _localEventBus = localEventBus;
 }

 public virtual async Task ChangeStockCountAsync(Guid productId, int newCount)
 {
 //TODO: IMPLEMENT YOUR LOGIC...

 //PUBLISH THE EVENT
 await _localEventBus.PublishAsync(
 new StockCountChangedEvent
 {
 ProductId = productId,
 NewCount = newCount
 }
);
 }
 }
}...```

```

`'PublishAsync'` method gets a single parameter: the event object, which is responsible to hold the data related to the event. It is a simple plain class:

```
```csharp
using System;

namespace AbpDemo
{
    public class StockCountChangedEvent
    {
        public Guid ProductId { get; set; }
```

```

        public int NewCount { get; set; }
    }
}...

```

Even if you don't need to transfer any data, you need to create a class (which is an empty class in this case).

Publishing Events Inside Entity / Aggregate Root Classes

[Entities](Entities.md) can not inject services via dependency injection, but it is very common to publish local events inside entity / aggregate root classes.

****Example: Publish a local event inside an aggregate root method****

```

```csharp
using System;
using Volo.Abp.Domain.Entities;

namespace AbpDemo
{
 public class Product : AggregateRoot<Guid>
 {
 public string Name { get; set; }

 public int StockCount { get; private set; }

 private Product() { }

 public Product(Guid id, string name)
 : base(id)
 {
 Name = name;
 }

 public void ChangeStockCount(int newCount)
 {
 StockCount = newCount;

 //ADD an EVENT TO BE PUBLISHED
 AddLocalEvent(
 new StockCountChangedEvent
 {
 ProductId = Id,
 NewCount = newCount
 }
);
 }
 }
}...

```

'AggregateRoot' class defines the 'AddLocalEvent' to add a new local event, that is published when the aggregate root object is saved (created, updated or deleted) into the database.

> Tip: If an entity publishes such an event, it is a good practice to change the related properties in a controlled manner, just like the example above - `StockCount` can only be changed by the `ChangeStockCount` method which guarantees publishing the event.

#### #### IGeneratesDomainEvents Interface

Actually, adding local events are not unique to the `AggregateRoot` class. You can implement `IGeneratesDomainEvents` for any entity class. But, `AggregateRoot` implements it by default and makes it easy for you.

> It is not suggested to implement this interface for entities those are not aggregate roots, since it may not work for some database providers for such entities. It works for EF Core, but not works for MongoDB for example.

#### #### How It Was Implemented?

Calling the `AddLocalEvent` doesn't immediately publish the event. The event is published when you save changes to the database;

- \* For EF Core, it is published on `DbContext.SaveChanges`.
- \* For MongoDB, it is published when you call repository's `InsertAsync`, `UpdateAsync` or `DeleteAsync` methods (since MongoDB has not a change tracking system).

#### ## Subscribing to Events

A service can implement the `ILocalEventHandler<TEvent>` to handle the event.

\*\*Example: Handle the `StockCountChangedEvent` defined above\*\*

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.EventBus;

namespace AbpDemo
{
    public class MyHandler
        : ILocalEventHandler<StockCountChangedEvent>,
        ITransientDependency
    {
        public async Task HandleEventAsync(StockCountChangedEvent eventData)
        {
            //TODO: your code that does something on the event
        }
    }
}...```

```

That's all. `MyHandler` is **automatically discovered** by the ABP Framework and `HandleEventAsync` is called whenever a `StockCountChangedEvent` occurs. You can inject any service and perform any required logic in your handler class.

- * **One or more handlers** can subscribe to the same event.
- * A single event handler class can **subscribe to multiple events** by implementing the `ILocalEventHandler<TEvent>` interface for each event type.

If you perform **database operations** and use the [repositories](Repositories.md) inside the event handler, you may need to create a [unit of work](Unit-Of-Work.md), because some repository methods need to work inside an **active unit of work**. Make the handle method `virtual` and add a `[UnitOfWork]` attribute for the method, or manually use the `IUnitOfWorkManager` to create a unit of work scope.

> The handler class must be registered to the dependency injection (DI). The sample above uses the `ITransientDependency` to accomplish it. See the [DI document](Dependency-Injection.md) for more options.

LocalEventHandlerOrder Attribute

`LocalEventHandlerOrder` attribute can be used to set the execution order for the event handlers, which can be helpful if you want to handle your event handlers in a specific order.

```
```csharp
[LocalEventHandlerOrder(-1)]
public class MyHandler
 : ILocalEventHandler<StockCountChangedEvent>,
 ITransientDependency
{
 public async Task HandleEventAsync(StockCountChangedEvent eventData)
 {
 //TODO: your code that does something on the event
 }
}
...```

```

> By default, all event handlers have an order value of 0. Thus, if you want to take certain event handlers to be executed before other event handlers, you can set the order value as a negative value.

### #### LocalEventHandlerOrderAttribute Properties

- \* `Order` (`int`): Used to set the execution order for a certain event handler.

### ### Transaction & Exception Behavior

Event handlers are always executed in the same [unit of work](Unit-Of-Work.md) scope, that means in the same database transaction with the code that published the event. If an event handler throws an exception, the unit of work (database transaction) is rolled back. So, \*\*use try-catch yourself\*\* in the event handler if you want to hide the error.

When you call `ILocalEventBus.PublishAsync`, the event handlers are not immediately executed. Instead, they are executed just before the current unit of work completed (an unhandled exception in the handler still rollbacks the current unit of work). If you want to immediately execute the handlers, set the optional `onUnitOfWorkComplete` parameter to `false`.

> Keeping the default behavior is recommended unless you don't have a unique requirement. `onUnitOfWorkComplete` option is not available when you publish events inside entity / aggregate root classes (see the \*Publishing Events Inside Entity / Aggregate Root Classes\* section).

## ## Pre-Built Events

It is very common to \*\*publish events on entity create, update and delete\*\* operations. ABP Framework \*\*automatically\*\* publish these events for all entities. You can just subscribe to the related event.

\*\*Example: Subscribe to an event that published when a user was created\*\*

```
```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Identity;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Entities.Events;
using Volo.Abp.EventBus;

namespace AbpDemo
{
    public class MyHandler
        : ILocalEventHandler<EntityCreatedEventData<IdentityUser>>,
        ITransientDependency
    {
        public async Task HandleEventAsync(
            EntityCreatedEventData<IdentityUser> eventData)
        {
            var userName = eventData.Entity.UserName;
            var email = eventData.Entity.Email;
            //...
        }
    }
}...```

```

This class subscribes to the `EntityCreatedEventData<IdentityUser>` , which is published just after a user was created (but before the current transaction is completed). For example, you may want to send a "Welcome" email to the new user.

The pre-built event types are;

- * `EntityCreatedEventData<T>` is published just after an entity was successfully created.
- * `EntityUpdatedEventData<T>` is published just after an entity was successfully updated.
- * `EntityDeletedEventData<T>` is published just after an entity was successfully deleted.
- * `EntityChangedEventArgs<T>` is published just after an entity was successfully created, updated or deleted. It can be a shortcut if you need to listen any type of change - instead of subscribing to the individual events.

How It Was Implemented?

Pre-build events are published when you save changes to the database;

- * For EF Core, they are published on `DbContext.SaveChanges` .
- * For MongoDB, they are published when you call repository's `InsertAsync` , `UpdateAsync` or `DeleteAsync` methods (since MongoDB has not a change tracking system).

See Also

- * [Distributed Event Bus](Distributed-Event-Bus.md)

7.13.3 Distributed Event Bus

Distributed Event Bus

Distributed Event bus system allows to **publish** and **subscribe** to events that can be **transferred across application/service boundaries**. You can use the distributed event bus to asynchronously send and receive messages between **microservices** or **applications**.

Providers

Distributed event bus system provides an **abstraction** that can be implemented by any vendor/provider. There are four providers implemented out of the box:

- * `LocalDistributedEventBus` is the default implementation that implements the distributed event bus to work as in-process. Yes! The **default implementation works just like the [local event bus](Local-Event-Bus.md)**, if you don't configure a real distributed provider.
- * `AzureDistributedEventBus` implements the distributed event bus with the [Azure Service Bus](https://azure.microsoft.com/en-us/services/service-bus/). See the [Azure Service Bus integration document](Distributed-Event-Bus-Azure-Integration.md) to learn how to configure it.
- * `RabbitMqDistributedEventBus` implements the distributed event bus with the [RabbitMQ](https://www.rabbitmq.com/). See the [RabbitMQ integration document](Distributed-Event-Bus-RabbitMQ-Integration.md) to learn how to configure it.
- * `KafkaDistributedEventBus` implements the distributed event bus with the [Kafka](https://kafka.apache.org/). See the [Kafka integration document](Distributed-Event-Bus-Kafka-Integration.md) to learn how to configure it.
- * `RebusDistributedEventBus` implements the distributed event bus with the [Rebus](http://mookid.dk/category/rebus/). See the [Rebus integration document](Distributed-Event-Bus-Rebus-Integration.md) to learn how to configure it.

Using a local event bus as default has a few important advantages. The most important one is that: It allows you to write your code compatible to distributed architecture. You can write a monolithic application now that can be split into microservices later. It is a good practice to communicate between bounded contexts (or between application modules) via distributed events instead of local events.

For example, [pre-built application modules](Modules/Index.md) is designed to work as a service in a distributed system while they can also work as a module in a monolithic application without depending on an external message broker.

Publishing Events

There are two ways of publishing distributed events explained in the following sections.

Using `IDistributedEventBus` to Publish Events

`'IDistributedEventBus'` can be [injected](Dependency-Injection.md) and used to publish a distributed event.

****Example: Publish a distributed event when the stock count of a product changes****

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.EventBus.Distributed;

namespace AbpDemo
{
 public class MyService : ITransientDependency
 {
 private readonly IDistributedEventBus _distributedEventBus;

 public MyService(IDistributedEventBus distributedEventBus)
 {
 _distributedEventBus = distributedEventBus;
 }

 public virtual async Task ChangeStockCountAsync(Guid productId, int newCount)
 {
 await _distributedEventBus.PublishAsync(
 new StockCountChangedEto
 {
 ProductId = productId,
 NewCount = newCount
 }
);
 }
 }
}...```

```

`'PublishAsync'` method gets the event object, which is responsible to hold the data related to the event. It is a simple plain class:

```
```csharp
using System;

namespace AbpDemo
{
    [EventName("MyApp.Product.StockChange")]
    public class StockCountChangedEto
    {
        public Guid ProductId { get; set; }

        public int NewCount { get; set; }
    }
}```
```

```
} ...
```

Even if you don't need to transfer any data, you need to create a class (which is an empty class in this case).

> `Eto` is a suffix for **Event**Transfer **O**bjects we use by convention. While it is not required, we find it useful to identify such event classes (just like [DTOs](Data-Transfer-Objects.md) on the application layer).

Event Name

`EventName` attribute is optional, but suggested. If you don't declare it for an event type (ETO class), the event name will be the full name of the event class, `AbpDemo.StockCountChangedEto` in this case.

About Serialization for the Event Objects

Event transfer objects (ETOs) **must be serializable** since they will be serialized/deserialized to JSON or other format when it is transferred to out of the process.

Avoid circular references, polymorphism, private setters and provide default (empty) constructors if you have any other constructor as a good practice (while some serializers may tolerate it), just like the DTOs.

Publishing Events Inside Entity / Aggregate Root Classes

[Entities](Entities.md) can not inject services via dependency injection, but it is very common to publish distributed events inside entity / aggregate root classes.

Example: Publish a distributed event inside an aggregate root method

```
```csharp
using System;
using Volo.Abp.Domain.Entities;

namespace AbpDemo
{
 public class Product : AggregateRoot<Guid>
 {
 public string Name { get; set; }

 public int StockCount { get; private set; }

 private Product() { }

 public Product(Guid id, string name)
 : base(id)
 {
 Name = name;
 }

 public void ChangeStockCount(int newCount)
 {
 StockCount = newCount;
 }
}
```

```

 //ADD an EVENT TO BE PUBLISHED
 AddDistributedEvent(
 new StockCountChangedEto
 {
 ProductId = Id,
 NewCount = newCount
 }
);
 }
}
```

```

`'AggregateRoot'` class defines the `'AddDistributedEvent'` to add a new distributed event, that is published when the aggregate root object is saved (created, updated or deleted) into the database.

> If an entity publishes such an event, it is a good practice to change the related properties in a controlled manner, just like the example above - `'StockCount'` can only be changed by the `'ChangeStockCount'` method which guarantees publishing the event.

IGeneratesDomainEvents Interface

Actually, adding distributed events are not unique to the `'AggregateRoot'` class. You can implement `'IGeneratesDomainEvents'` for any entity class. But, `'AggregateRoot'` implements it by default and makes it easy for you.

> It is not suggested to implement this interface for entities those are not aggregate roots, since it may not work for some database providers for such entities. It works for EF Core, but not works for MongoDB for example.

How It Was Implemented?

Calling the `'AddDistributedEvent'` doesn't immediately publish the event. The event is published when you save changes to the database;

- * For EF Core, it is published on `'DbContext.SaveChanges'`.
- * For MongoDB, it is published when you call repository's `'InsertAsync'`, `'UpdateAsync'` or `'DeleteAsync'` methods (since MongoDB has not a change tracking system).

Subscribing to Events

A service can implement the `'IDistributedEventHandler<TEvent>'` to handle the event.

Example: Handle the `'StockCountChangedEto'` defined above

```

```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.EventBus.Distributed;

namespace AbpDemo
{
 public class MyHandler

```

```

 : IDistributedEventHandler<StockCountChangedEto>,
 ITransientDependency
{
 public async Task HandleEventAsync(StockCountChangedEto eventData)
 {
 var productId = eventData.ProductId;
 }
}
}...

```

That's all.

- \* `MyHandler` is \*\*automatically discovered\*\* by the ABP Framework and `HandleEventAsync` is called whenever a `StockCountChangedEto` event occurs.
- \* If you are using a distributed message broker, like RabbitMQ, ABP automatically \*\*subscribes to the event on the message broker\*\*, gets the message, executes the handler.
- \* It sends \*\*confirmation (ACK)\*\* to the message broker if the event handler was successfully executed (did not throw any exception).

You can inject any service and perform any required logic here. A single event handler class can \*\*subscribe to multiple events\*\* but implementing the `IDistributedEventHandler<TEvent>` interface for each event type.

If you perform \*\*database operations\*\* and use the [repositories](Repositories.md) inside the event handler, you may need to create a [unit of work](Unit-Of-Work.md), because some repository methods need to work inside an \*\*active unit of work\*\*. Make the handle method `virtual` and add a `[UnitOfWork]` attribute for the method, or manually use the `IUnitOfWorkManager` to create a unit of work scope.

> The handler class must be registered to the dependency injection (DI). The sample above uses the `ITransientDependency` to accomplish it. See the [DI document](Dependency-Injection.md) for more options.

## ## Monitoring Distributed Events

The ABP Framework allows you to stay informed when your application \*\*receives\*\* or \*\*sends\*\* a distributed event. This capability enables you to track the event flow within your application and take appropriate actions based on the received or sent distributed events.

### ### Received Events

The `DistributedEventReceived` local event is published when your application receives an event from the distributed event bus. `DistributedEventReceived` class has the following fields:

- \*\*`Source`\*\*: It represents the source of the distributed event. Source can be `Direct`, `Inbox`, `Outbox`.
- \*\*`EventName`\*\*: It represents the [name](#event-name) of the event received.
- \*\*`EventData`\*\*: It represents the actual data associated with the event received. Since it is of type `object`, it can hold any type of data.

**\*\*Example: Get informed when your application receives an event from the distributed event bus\*\***

```

```csharp
public class DistributedEventReceivedHandler : 
ILocalEventHandler<DistributedEventReceived>, ITransientDependency
{
    public async Task HandleEventAsync(DistributedEventReceived eventData)
    {
        // TODO: IMPLEMENT YOUR LOGIC...
    }
}
```

```

### ### Sent Events

The '`DistributedEventSent`' local event is published when your application sends an event to the distributed event bus. '`DistributedEventSent`' class has the following fields:

- \*\*`Source`\*\*: It represents the source of the distributed event. Source can be '`Direct`', '`Inbox`', '`Outbox`'.
- \*\*`EventName`\*\*: It represents the [name](#event-name) of the event sent.
- \*\*`EventData`\*\*: It represents the actual data associated with the event sent. Since it is of type '`object`', it can hold any type of data.

**\*\*Example: Get informed when your application sends an event to the distributed event bus\*\***

```

```csharp
public class DistributedEventSentHandler : 
ILocalEventHandler<DistributedEventSent>, ITransientDependency
{
    public async Task HandleEventAsync(DistributedEventSent eventData)
    {
        // TODO: IMPLEMENT YOUR LOGIC...
    }
}
```

```

You can seamlessly integrate event-tracking capabilities into your application by subscribing to the '`DistributedEventReceived`' and '`DistributedEventSent`' local events as above examples. This empowers you to effectively monitor the messaging flow, diagnose any potential issues, and gain valuable insights into the behavior of your distributed messaging system.

### ## Pre-Defined Events

ABP Framework **automatically publishes** distributed events for **create, update and delete** operations for an [entity](Entities.md) once you configure it.

### ### Event Types

There are three pre-defined event types:

- \* '`EntityCreatedEto<T>`' is published when an entity of type '`T`' was created.
- \* '`EntityUpdatedEto<T>`' is published when an entity of type '`T`' was updated.
- \* '`EntityDeletedEto<T>`' is published when an entity of type '`T`' was deleted.

These types are generics. 'T' is actually the type of the \*\*Event\*\* rather than the type of the entity. Because, an entity object can not be transferred as a part of the event data. So, it is typical to define a ETO class for an entity class, like 'ProductEto' for 'Product' entity.

### ### Subscribing to the Events

Subscribing to the auto events is same as subscribing a regular distributed event.

\*\*Example: Get notified once a product updated\*\*

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Entities.Events.Distributed;
using Volo.Abp.EventBus.Distributed;

namespace AbpDemo
{
    public class MyHandler :
        IDistributedEventHandler<EntityUpdatedEto<ProductEto>>,
        ITransientDependency
    {
        public async Task HandleEventAsync(EntityUpdatedEto<ProductEto>
eventData)
        {
            var productId = eventData.Entity.Id;
            //TODO
        }
    }
}

* `MyHandler` implements the
`IDistributedEventHandler<EntityUpdatedEto<ProductEto>>`.
* It is required to register your handler class to the [dependency
injection](Dependency-Injection.md) system. Implementing
`ITransientDependency` like in this example is an easy way.
```

Configuration

You can configure the 'AbpDistributedEntityEventOptions' in the 'ConfigureServices' of your [module](Module-Development-Basics.md) to add a selector.

Example: Configuration samples

```
```csharp
Configure<AbpDistributedEntityEventOptions>(options =>
{
 //Enable for all entities
 options.AutoEventSelectors.AddAll();

 //Enable for a single entity
 options.AutoEventSelectors.Add<Product>();
```

```

//Enable for all entities in a namespace (and child namespaces)
options.AutoEventSelectors.AddNamespace("MyProject.Products");

//Custom predicate expression that should return true to select a type
options.AutoEventSelectors.Add(
 type => type.Namespace.StartsWith("MyProject."))
);
}...

```

\* The last one provides flexibility to decide if the events should be published for the given entity type. Returns 'true' to accept a 'Type'.

You can add more than one selector. If one of the selectors match for an entity type, then it is selected.

### ### Event Transfer Object

Once you enable \*\*auto events\*\* for an entity, ABP Framework starts to publish events on the changes on this entity. If you don't specify a corresponding \*\*Event Transfer Object (ETO)\*\* for the entity, ABP Framework uses a standard type, named '`EntityEto`', which has only two properties:

- \* `'EntityType'` (`'string'`): Full name (including namespace) of the entity class.
- \* `'KeysAsString'` (`'string'`): Primary key(s) of the changed entity. If it has a single key, this property will be the primary key value. For a composite key, it will contain all keys separated by ',' (comma).

So, you can implement the `'IDistributedEventHandler<EntityUpdatedEto<EntityEto>'` to subscribe the update events. However, it is not a good approach to subscribe to such a generic event, because you handle the update events for all entities in a single handler (since they all use the same ETO object). You can define the corresponding ETO type for the entity type.

**\*\*Example: Declare to use '`ProductEto`' for the '`Product`' entity\*\***

```

```csharp
public class ProductEto
{
    public Guid Id { get; set; }
    public string Name { get; set; }
    public float Price { get; set; }
}
```

```

Then you can use the '`AbpDistributedEntityEventOptions.EtoMappings`' option to map your '`Product`' entity to the '`ProductEto`':

```

```csharp
Configure<AbpDistributedEntityEventOptions>(options =>
{
    options.AutoEventSelectors.Add<Product>();
    options.EtoMappings.Add<Product, ProductEto>();
});
```

```

....

This example;

```
* Adds a selector to allow to publish the create, update and delete events
for the 'Product' entity.
* Configure to use the 'ProductEto' as the event transfer object to publish
for the 'Product' related events.

> Distributed event system use the [object to object mapping](Object-To-
Object-Mapping.md) system to map 'Product' objects to 'ProductEto' objects.
So, you need to configure the object mapping ('Product' -> 'ProductEto') too.
You can check the [object to object mapping document](Object-To-Object-
Mapping.md) to learn how to do it.
```

## ## Entity Synchronizer

In a distributed (or microservice) system, it is typical to subscribe to change events for an [entity](Entities.md) type of another service, so you can get notifications when the subscribed entity changes. In that case, you can use ABP's Pre-Defined Events as explained in the previous section.

If your purpose is to store your local copies of a remote entity, you typically subscribe to create, update and delete events of the remote entity and update your local database in your event handler. ABP provides a pre-built 'EntitySynchronizer' base class to make that operation easier for you.

Assume that there is a 'Product' entity (probably an aggregate root entity) in a Catalog microservice, and you want to keep copies of the products in your Ordering microservice, with a local 'OrderProduct' entity. In practice, properties of the 'OrderProduct' class will be a subset of the 'Product' properties, because not all the product data is needed in the Ordering microservice (however, you can make a full copy if you need). Also, the 'OrderProduct' entity may have additional properties that are populated and used in the Ordering microservice.

The first step to establish the synchronization is to define an ETO (Event Transfer Object) class in the Catalog microservice that is used to transfer the event data. Assuming the 'Product' entity has a 'Guid' key, your ETO can be as shown below:

```
```csharp  
[EventName("product")]  
public class ProductEto : EntityEto<Guid>  
{  
    // Your Product properties here...  
}  
```
```

'ProductEto' can be put in a shared project (DLL) that is referenced by the Catalog and the Ordering microservices. Alternatively, you can put a copy of the 'ProductEto' class in the Ordering microservice if you don't want to introduce a common project dependency between the services. In this case, the 'EventName' attribute becomes critical to map the 'ProductEto' classes across two services (you should use the same event name).

Once you define an ETO class, you should configure the ABP Framework to publish auto (create, update and delete) events for the '[Product](#)' entity, as explained in the previous section:

```
```csharp
Configure<AbpDistributedEntityEventOptions>(options =>
{
    options.AutoEventSelectors.Add<Product>();
    options.EtoMappings.Add<Product, ProductEto>();
});
...```

```

Finally, you should create a class in the Ordering microservice, that is derived from the '[EntitySynchronizer](#)' class:

```
```csharp
public class ProductSynchronizer : EntitySynchronizer<OrderProduct, Guid,
ProductEto>
{
 public ProductSynchronizer(
 IObjectMapper objectMapper,
 IRepository<OrderProduct, Guid> repository
) : base(objectMapper, repository)
 {
 }
}
...```

```

The main point of this class is it subscribes to the create, update and delete events of the source entity and updates the local entity in the database. It uses the [\[Object Mapper\]\(Object-To-Object-Mapping.md\)](#) system to create or update the '[OrderProduct](#)' objects from the '[ProductEto](#)' objects. So, you should also configure the object mapper to make it properly work. Otherwise, you should manually perform the object mapping by overriding the '[`MapToEntityAsync\(TSourceEntityEto\)`](#)' and '[`MapToEntityAsync\(TSourceEntityEto, TEntity\)`](#)' methods in your '[`ProductSynchronizer`](#)' class.

If your entity has a composite primary key (see the [\[Entities document\]\(Entities.md\)](#)), then you should inherit from the '[`EntitySynchronizer< TEntity, TSourceEntityEto >`](#)' class (just don't use the '[`Guid`](#)' generic argument in the previous example) and implement '[`FindLocalEntityAsync`](#)' to find the entity in your local database using the '[`Repository`](#)'.

'[EntitySynchronizer](#)' is compatible with the *\*Entity Versioning\** system (see the [\[Entities document\]\(Entities.md\)](#)). So, it works as expected even if the events are received as disordered. If the entity's version in your local database is newer than the entity in the received event, then the event is ignored. You should implement the '[`IHasEntityVersion`](#)' interface for the entity and ETO classes (for this example, you should implement for the '[`Product`](#)', '[`ProductEto`](#)' and '[`OrderProduct`](#)' classes).

If you want to ignore some type of change events, you can set '[`IgnoreEntityCreatedEvent`](#)', '[`IgnoreEntityUpdatedEvent`](#)' and '[`IgnoreEntityDeletedEvent`](#)' in the constructor of your class. Example:

```
```csharp
...```

```

```

public class ProductSynchronizer
    : EntitySynchronizer<OrderProduct, Guid, ProductEto>
{
    public ProductSynchronizer(
        IObjectMapper objectMapper,
        IRepository<OrderProduct, Guid> repository
    ) : base(objectMapper, repository)
    {
        IgnoreEntityDeletedEvent = true;
    }
}
...

```

> Notice that the `EntitySynchronizer` can only create/update the entities after you use it. If you have an existing system with existing data, you should manually copy the data for one time, because the `EntitySynchronizer` starts to work.

Transaction and Exception Handling

Distributed event bus works in-process (since default implementation is `LocalDistributedEventBus`) unless you configure an actual provider (e.g. [Kafka](Distributed-Event-Bus-Kafka-Integration.md) or [RabbitMQ](Distributed-Event-Bus-RabbitMQ-Integration.md)). In-process event bus always executes event handlers in the same [unit of work](Unit-Of-Work.md) scope that you publishes the events in. That means, if an event handler throws an exception, then the related unit of work (the database transaction) is rolled back. In this way, your application logic and event handling logic becomes transactional (atomic) and consistent. If you want to ignore errors in an event handler, you must use a `try-catch` block in your handler and shouldn't re-throw the exception.

When you switch to an actual distributed event bus provider (e.g. [Kafka](Distributed-Event-Bus-Kafka-Integration.md) or [RabbitMQ](Distributed-Event-Bus-RabbitMQ-Integration.md)), then the event handlers will be executed in different processes/applications as their purpose is to create distributed systems. In this case, the only way to implement transactional event publishing is to use the outbox/inbox patterns as explained in the *Outbox / Inbox for Transactional Events* section.

If you don't configure outbox/inbox pattern or use the `LocalDistributedEventBus`, then events are published at the end of the unit of work by default, just before the unit of work is completed (that means throwing exception in an event handler still rollbacks the unit of work), even if you publish them in the middle of unit of work. If you want to immediately publish the event, you can set `onUnitOfWorkComplete` to `false` while using `IDistributedEventBus.PublishAsync` method.

> Keeping the default behavior is recommended unless you don't have a unique requirement. `onUnitOfWorkComplete` option is not available when you publish events inside entity / aggregate root classes (see the *Publishing Events Inside Entity / Aggregate Root Classes* section).

Outbox / Inbox for Transactional Events

The **[transactional outbox pattern](https://microservices.io/patterns/data/transactional-outbox.html)** is used to publishing distributed events within the **same transaction** that

manipulates the application's database. When you enable outbox, distributed events are saved into the database inside the same transaction with your data changes, then sent to the actual message broker by a separate [background worker](Background-Workers.md) with a re-try system. In this way, it ensures the consistency between your database state and the published events.

The **transactional inbox pattern**, on the other hand, saves incoming events into database first. Then (in a [background worker](Background-Workers.md)) executes the event handler in a transactional manner and removes the event from the inbox queue in the same transaction. It ensures that the event is only executed one time by keeping the processed messages for a while and discarding the duplicate events received from the message broker.

Enabling the event outbox and inbox systems require a few manual steps for your application. Please apply the instructions in the following sections to make them running.

> Outbox and Inbox can be separately enabled and configured, so you may only use one of them if you want.

Pre-requirements

- * The outbox/inbox system uses the distributed lock system to handle concurrency when you run multiple instances of your application/service. So, you should **configure the distributed lock system** with one of the providers as [explained in this document](Distributed-Locking.md).
 - * The outbox/inbox system supports [Entity Framework Core](Entity-Framework-Core.md) (EF Core) and [MongoDB](MongoDB.md) **database providers** out of the box. So, your applications should use one of these database providers. For other database providers, see the **Implementing a Custom Database Provider** section.
- > If you are using MongoDB, be sure that you enabled multi-document database transactions that was introduced in MongoDB version 4.0. See the **Transactions** section of the [MongoDB](MongoDB.md) document.

Enabling event outbox

Open your `DbContext` class (EF Core or MongoDB), implement the `IHasEventOutbox` interface. You should end up by adding a `DbSet` property into your `DbContext` class:

```
```csharp
public DbSet<OutgoingEventRecord> OutgoingEvents { get; set; }
````
```

Add the following lines inside the `OnModelCreating` method of your `DbContext` class (only for EF Core):

```
```csharp
builder.ConfigureEventOutbox();
````
```

For EF Core, use the standard `Add-Migration` and `Update-Database` commands to apply changes into your database (you can skip this step for MongoDB). If you want to use the command-line terminal, run the following commands in the root directory of the database integration project:

```
```bash
dotnet ef migrations add "Added_Event_Outbox"
dotnet ef database update
````
```

Finally, write the following configuration code inside the `'ConfigureServices'` method of your [module class] (Module-Development-Basics.md) (replace `'YourDbContext'` with your own `'DbContext'` class):

```
```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
 options.Outboxes.Configure(config =>
 {
 config.UseDbContext<YourDbContext>();
 });
});
```

> \*\*IMPORTANT\*\*: Outbox sending service uses distributed locks to ensure only a single instance of your application consumes the outbox queue concurrently. Distributed locking key should be unique per database. The `'config'` object (in the preceding code example) has a `'DatabaseName'` property, which is used in the distributed lock key to ensure the uniqueness. `'DatabaseName'` is automatically set by the `'UseDbContext'` method, getting the database name from the `'ConnectionStringName'` attribute of the `'YourDbContext'` class. So, if you have multiple databases in your system, ensure that you use the same connection string name for the same database, but different connection string names for different databases. If you can't ensure that, you can manually set `'config.DatabaseName'` (after the `'UseDbContext'` line) to ensure that uniqueness.

### ### Enabling event inbox

Open your `'DbContext'` class (EF Core or MongoDB), implement the `'IHasEventInbox'` interface. You should end up by adding a `'DbSet'` property into your `'DbContext'` class:

```
```csharp
public DbSet<IncomingEventRecord> IncomingEvents { get; set; }
````
```

Add the following lines inside the `'OnModelCreating'` method of your `'DbContext'` class (only for EF Core):

```
```csharp
builder.ConfigureEventInbox();
````
```

For EF Core, use the standard `'Add-Migration'` and `'Update-Database'` commands to apply changes into your database (you can skip this step for MongoDB). If you want to use the command-line terminal, run the following commands in the root directory of the database integration project:

```
```bash
dotnet ef migrations add "Added_Event_Inbox"
dotnet ef database update
````
```

Finally, write the following configuration code inside the `'ConfigureServices'` method of your [module class] (Module-Development-Basics.md) (replace `'YourDbContext'` with your own `'DbContext'` class):

```
```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
    options.Inboxes.Configure(config =>
    {
        config.UseDbContext<YourDbContext>();
    });
});
```

****IMPORTANT**:** Inbox processing service uses distributed locks to ensure only a single instance of your application consumes the inbox queue concurrently. Distributed locking key should be unique per database. The `'config'` object (in the preceding code example) has a `'DatabaseName'` property, which is used in the distributed lock key to ensure the uniqueness. `'DatabaseName'` is automatically set by the `'UseDbContext'` method, getting the database name from the `'ConnectionStringName'` attribute of the `'YourDbContext'` class. So, if you have multiple databases in your system, ensure that you use the same connection string name for the same database, but different connection string names for different databases. If you can't ensure that, you can manually set `'config.DatabaseName'` (after the `'UseDbContext'` line) to ensure that uniqueness.

Additional Configuration

> The default configuration will be enough for most cases. However, there are some options you may want to set for outbox and inbox.

Outbox configuration

Remember how outboxes are configured:

```
```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
 options.Outboxes.Configure(config =>
 {
 // TODO: Set options
 });
});
```

Here, the following properties are available on the `'config'` object:

- \* `'IsSendingEnabled'` (default: `'true'`): You can set to `'false'` to disable sending outbox events to the actual event bus. If you disable this, events can still be added to outbox, but not sent. This can be helpful if you have multiple applications (or application instances) writing to outbox, but use one of them to send the events.
- \* `'Selector'`: A predicate to filter the event (ETO) types to be used for this configuration. Should return `'true'` to select the event. It selects all the events by default. This is especially useful if you want to ignore some ETO types from the outbox, or want to define named outbox configurations and

group events within these configurations. See the *\*Named Configurations\** section.

- \* **ImplementationType**: Type of the class that implements the database operations for the outbox. This is normally set when you call `'UseDbContext'` as shown before. See *\*Implementing a Custom Outbox/Inbox Database Provider\** section for advanced usages.
- \* **DatabaseName**: Unique database name for the database that is used for this outbox configuration. See the **\*\*IMPORTANT\*\*** paragraph at the end of the *\*Enabling event outbox\** section.

#### **#### Inbox configuration**

Remember how inboxes are configured:

```
```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
    options.Inboxes.Configure(config =>
    {
        // TODO: Set options
    });
});
```

Here, the following properties are available on the `'config'` object:

- * **IsProcessingEnabled** (default: `'true'`): You can set to `'false'` to disable processing (handling) events in the inbox. If you disable this, events can still be received, but not executed. This can be helpful if you have multiple applications (or application instances), but use one of them to execute the event handlers.
- * **EventSelector**: A predicate to filter the event (ETO) types to be used for this configuration. This is especially useful if you want to ignore some ETO types from the inbox, or want to define named inbox configurations and group events within these configurations. See the **Named Configurations** section.
- * **HandlerSelector**: A predicate to filter the event handled types (classes implementing the `'IDistributedEventHandler<TEvent>'` interface) to be used for this configuration. This is especially useful if you want to ignore some event handler types from inbox processing, or want to define named inbox configurations and group event handlers within these configurations. See the **Named Configurations** section.
- * **ImplementationType**: Type of the class that implements the database operations for the inbox. This is normally set when you call `'UseDbContext'` as shown before. See **Implementing a Custom Outbox/Inbox Database Provider** section for advanced usages.
- * **DatabaseName**: Unique database name for the database that is used for this outbox configuration. See the ****IMPORTANT**** paragraph at the end of the **Enabling event inbox** section.

AbpEventBusBoxesOptions

`'AbpEventBusBoxesOptions'` can be used to fine-tune how inbox and outbox systems work. For most of the systems, using the defaults would be more than enough, but you can configure it to optimize your system when it is needed.

Just like all the `[options classes](Options.md)`, `'AbpEventBusBoxesOptions'` can be configured in the `'ConfigureServices'` method of your `[module class](Module-Development-Basics.md)` as shown in the following code block:

```
```csharp
Configure<AbpEventBusBoxesOptions>(options =>
{
 // TODO: configure the options
});
```

```

`AbpEventBusBoxesOptions` has the following properties to be configured:

- * `BatchPublishOutboxEvents`: Can be used to enable or disable batch publishing events to the message broker. Batch publishing works if it is supported by the distributed event bus provider. If not supported, events are sent one by one as the fallback logic. Keep it as enabled since it has a great performance gain wherever possible. Default value is `true` (enabled).
- * `PeriodTimeSpan`: The period of the inbox and outbox message processors to check if there is a new event in the database. Default value is 2 seconds (`TimeSpan.FromSeconds(2)`).
- * `CleanOldEventTimeIntervalSpan`: The event inbox system periodically checks and deletes the old processed events from the inbox in the database. You can set this value to determine the check period. Default value is 6 hours (`TimeSpan.FromHours(6)`).
- * `WaitTimeToDeleteProcessedInboxEvents`: Inbox events are not deleted from the database for a while even if they are successfully processed. This is for a system to prevent multiple process of the same event (if the event broker sends it twice). This configuration value determines the time to keep the processed events. Default value is 2 hours (`TimeSpan.FromHours(2)`).
- * `InboxWaitingEventMaxCount`: The maximum number of events to query at once from the inbox in the database. Default value is 1000.
- * `OutboxWaitingEventMaxCount`: The maximum number of events to query at once from the outbox in the database. Default value is 1000.
- * `DistributedLockWaitDuration`: ABP uses [distributed locking](Distributed-Locking.md) to prevent concurrent access to the inbox and outbox messages in the database, when running multiple instance of the same application. If an instance of the application can not obtain the lock, it tries after a duration. This is the configuration of that duration. Default value is 15 seconds (`TimeSpan.FromSeconds(15)`).

Skipping Outbox

`IDistributedEventBus.PublishAsync` method provides an optional parameter, `useOutbox`, which is set to `true` by default. If you bypass outbox and immediately publish an event, you can set it to `false` for a specific event publishing operation.

Advanced Topics

Named Configurations

> All the concepts explained in this section is also valid for inbox configurations. We will show examples only for outbox to keep the document shorter.

See the following outbox configuration code:

```
```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
```

```

```

        options.Outboxes.Configure(config =>
    {
        //TODO
    });
}};

```

```

This is equivalent of the following code:

```

```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
    options.Outboxes.Configure("Default", config =>
    {
        //TODO
    });
}};

```

```

'`Default`' is this code indicates the configuration name. If you don't specify it (like in the previous code block), '`Default`' is used as the configuration name.

That means you can define more than one configuration for outbox (also for inbox) with different names. ABP runs all the configured outboxes.

Multiple outboxes can be needed if your application have more than one database and you want to run different outbox queues for different databases. In this case, you can use the '`Selector`' option to decide the events should be handled by an outbox. See the *\*Additional Configurations\** section above.

#### #### Implementing a Custom Outbox/Inbox Database Provider

If your application or service is using a database provider other than [[EF Core](#)](Entity-Framework-Core.md) and [[MongoDB](#)](MongoDB.md), you should manually integrate outbox/inbox system with your database provider.

> Outbox and Inbox table/data must be stored in the same database with your application's data (since we want to create a single database transaction that includes application's database operations and outbox/inbox table operations). Otherwise, you should care about distributed (multi-database) transaction support which is not provided by most of the vendors and may require additional configuration.

ABP provides '`IEventOutbox`' and '`IEventInbox`' abstractions as extension point for the outbox/inbox system. You can create classes by implementing these interfaces and register them to [[dependency injection](#)](Dependency-Injection.md).

Once you implement your custom event boxes, you can configure '`AbpDistributedEventBusOptions`' to use your event box classes:

```

```csharp
Configure<AbpDistributedEventBusOptions>(options =>
{
    options.Outboxes.Configure(config =>
    {
        config.ImplementationType = typeof(MyOutbox); //Your Outbox class
    });
}};

```

```

```
});

options.Inboxes.Configure(config =>
{
 config.ImplementationType = typeof(MyInbox); //Your Inbox class
});
});

See Also
```

- \* [Local Event Bus](Local-Event-Bus.md)

#### 7.13.3.1 Azure Service Bus Integration

##### # Distributed Event Bus Azure Integration

> This document explains \*\*how to configure the [Azure Service Bus](https://azure.microsoft.com/en-us/services/service-bus/)\*\* as the distributed event bus provider. See the [distributed event bus document](Distributed-Event-Bus.md) to learn how to use the distributed event bus system

##### ## Installation

Use the ABP CLI to add [Volo.Abp.EventBus.Azure](https://www.nuget.org/packages/Volo.Abp.EventBus.Azure) NuGet package to your project:

- \* Install the [ABP CLI](https://docs.abp.io/en/abp/latest/CLI) if you haven't installed before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.EventBus.Azure` package.
- \* Run `abp add-package Volo.Abp.EventBus.Azure` command.

If you want to do it manually, install the [Volo.Abp.EventBus.Azure](https://www.nuget.org/packages/Volo.Abp.EventBus.Azure) NuGet package to your project and add `'[DependsOn(typeof(AbpEventBusAzureModule))]'` to the [ABP module](Module-Development-Basics.md) class inside your project.

##### ## Configuration

You can configure using the standard [configuration system](Configuration.md), like using the `appsettings.json` file, or using the [options](Options.md) classes.

##### ### `appsettings.json` file configuration

This is the simplest way to configure the Azure Service Bus settings. It is also very strong since you can use any other configuration source (like environment variables) that is [supported by the AspNet Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/).

**\*\*Example: The minimal configuration to connect to Azure Service Bus Namespace with default configurations\*\***

```
```json
{
  "Azure": {
    "ServiceBus": {
      "Connections": {
        "Default": {
          "ConnectionString": "Endpoint=sb://sb-my-
app.servicebus.windows.net/;SharedAccessKeyName={{{Policy
Name}}};SharedAccessKey={};EntityPath=marketing-consent"
        }
      }
    },
    "EventBus": {
      "ConnectionName": "Default",
      "SubscriberName": "MySubscriberName",
      "TopicName": "MyTopicName"
    }
  }
}...
```

```

- \* `MySubscriberName` is the name of this subscription, which is used as the **\*\*Subscriber\*\*** on the Azure Service Bus.
- \* `MyTopicName` is the **\*\*topic name\*\***.

See [[the Azure Service Bus document](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions)](<https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions>) to understand these options better.

#### #### Connections

If you need to connect to another Azure Service Bus Namespace the Default, you need to configure the connection properties.

**\*\*Example: Declare two connections and use one of them for the event bus\*\***

```
```json
{
  "Azure": {
    "ServiceBus": {
      "Connections": {
        "Default": {
          "ConnectionString": "Endpoint=sb://sb-my-
app.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;Sha
redAccessKey={{{SharedAccessKey}}}"
        },
        "SecondConnection": {
          "ConnectionString": "Endpoint=sb://sb-my-
app.servicebus.windows.net/;SharedAccessKeyName={{{Policy
Name}}};SharedAccessKey={{{SharedAccessKey}}}"
        }
      }
    },
    "EventBus": {
      "ConnectionName": "SecondConnection",
    }
  }
}...
```

```

```

 "SubscriberName": "MySubscriberName",
 "TopicName": "MyTopicName"
 }
}
...

```

This allows you to use multiple Azure Service Bus namespaces in your application, but select one of them for the event bus.

You can use any of the [\[ServiceBusAdministrationClientOptions\]](https://docs.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.administration.servicebusadministrationclientoptions?view=azure-dotnet)(<https://docs.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.administration.servicebusadministrationclientoptions?view=azure-dotnet>), [\[ServiceBusClientOptions\]](https://docs.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclientoptions?view=azure-dotnet)(<https://docs.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusclientoptions?view=azure-dotnet>), [\[ServiceBusProcessorOptions\]](https://docs.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessoroptions?view=azure-dotnet)(<https://docs.microsoft.com/en-us/dotnet/api/azure.messaging.servicebus.servicebusprocessoroptions?view=azure-dotnet>) properties for the connection.

**\*\*Example: Specify the Admin, Client and Processor options\*\***

```

```json
{
    "Azure": {
        "ServiceBus": {
            "Connections": {
                "Default": {
                    "ConnectionString": "Endpoint=sb://sb-my-
app.servicebus.windows.net/;SharedAccessKeyName={{{Policy
Name}}};SharedAccessKey={};" + EntityPath="marketing-consent",
                    "Admin": {
                        "Retry": {
                            "MaxRetries": 3
                        }
                    },
                    "Client": {
                        "RetryOptions": {
                            "MaxRetries": 1
                        }
                    },
                    "Processor": {
                        "AutoCompleteMessages": true,
                        "ReceiveMode": "ReceiveAndDelete"
                    }
                }
            }
        },
        "EventBus": {
            "ConnectionName": "Default",
            "SubscriberName": "MySubscriberName",
            "TopicName": "MyTopicName"
        }
    }
}
```

```

**### The Options Classes**

`'AbpAzureServiceBusOptions'` and `'AbpAzureEventBusOptions'` classes can be used to configure the connection strings and event bus options for Azure Service Bus.

You can configure this options inside the '[ConfigureServices](#)' of your [\[module\]\(Module-Development-Basics.md\)](#).

#### **\*\*Example: Configure the connection\*\***

```
```csharp
Configure<AbpAzureServiceBusOptions>(options =>
{
    options.Connections.Default.ConnectionString = "Endpoint=sb://sb-my-
app.servicebus.windows.net/;SharedAccessKeyName={{{Policy
Name}}};SharedAccessKey={}";
    options.Connections.Default.Admin.Retry.MaxRetries = 3;
    options.Connections.Default.Client.RetryOptions.MaxRetries = 1;
});
```

Using these options classes can be combined with the `'appsettings.json'` way. Configuring an option property in the code overrides the value in the configuration file.

7.13.3.2 RabbitMQ Integration

Distributed Event Bus RabbitMQ Integration

> This document explains **how to configure the [RabbitMQ](<https://www.rabbitmq.com/>)** as the distributed event bus provider. See the [distributed event bus document](Distributed-Event-Bus.md) to learn how to use the distributed event bus system

Installation

Use the ABP CLI to add [Volo.Abp.EventBus.RabbitMQ](<https://www.nuget.org/packages/Volo.Abp.EventBus.RabbitMQ>) NuGet package to your project:

- * Install the [ABP CLI](<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- * Open a command line (terminal) in the directory of the `'.csproj'` file you want to add the `'[Volo.Abp.EventBus.RabbitMQ](#)'` package.
- * Run `'[abp add-package Volo.Abp.EventBus.RabbitMQ](#)'` command.

If you want to do it manually, install the [Volo.Abp.EventBus.RabbitMQ](<https://www.nuget.org/packages/Volo.Abp.EventBus.RabbitMQ>) NuGet package to your project and add `'[\[DependsOn\(typeof\(AbpEventBusRabbitMqModule\)\)\]](#)'` to the [ABP module](Module-Development-Basics.md) class inside your project.

Configuration

You can configure using the standard [configuration system](Configuration.md), like using the `appsettings.json` file, or using the [options](Options.md) classes.

`appsettings.json` file configuration

This is the simplest way to configure the RabbitMQ settings. It is also very strong since you can use any other configuration source (like environment variables) that is [supported by the AspNet Core](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/).

Example: The minimal configuration to connect to a local RabbitMQ server with default configurations**

```
```json
{
 "RabbitMQ": {
 "EventBus": {
 "ClientName": "MyClientName",
 "ExchangeName": "MyExchangeName"
 }
 }
}```
```

\* `ClientName` is the name of this application, which is used as the \*\*queue name\*\* on the RabbitMQ.  
\* `ExchangeName` is the \*\*exchange name\*\*.

See [the RabbitMQ document](https://www.rabbitmq.com/dotnet-api-guide.html#exchanges-and-queues) to understand these options better.

#### #### Connections

If you need to connect to another server than the localhost, you need to configure the connection properties.

**Example: Specify the host name (as an IP address)\*\***

```
```json
{
  "RabbitMQ": {
    "Connections": {
      "Default": {
        "HostName": "123.123.123.123"
      }
    },
    "EventBus": {
      "ClientName": "MyClientName",
      "ExchangeName": "MyExchangeName"
    }
  }
}```
```

Defining multiple connections is allowed. In this case, you can specify the connection that is used for the event bus.

****Example: Declare two connections and use one of them for the event bus****

```
```json
{
 "RabbitMQ": {
 "Connections": {
 "Default": {
 "HostName": "123.123.123.123"
 },
 "SecondConnection": {
 "HostName": "321.321.321.321"
 }
 },
 "EventBus": {
 "ClientName": "MyClientName",
 "ExchangeName": "MyExchangeName",
 "ConnectionName": "SecondConnection"
 }
 }
}
````
```

This allows you to use multiple RabbitMQ server in your application, but select one of them for the event bus.

You can use any of the [ConnectionFactory](<http://rabbitmq.github.io/rabbitmq-dotnet-client/api/RabbitMQ.Client.ConnectionFactory.html#properties>) properties as the connection properties.

****Example: Specify the connection port****

```
```json
{
 "RabbitMQ": {
 "Connections": {
 "Default": {
 "HostName": "123.123.123.123",
 "Port": "5672"
 }
 }
 }
}
````
```

If you need to connect to the RabbitMQ cluster, you can use the `;` character to separate the host names.

****Example: Connect to the RabbitMQ cluster****

```
```json
{
 "RabbitMQ": {
 "Connections": {
 "Default": {
 "HostName": "123.123.123.123;234.234.234.234"
 }
 }
 }
}
````
```

```

        },
        "EventBus": {
            "ClientName": "MyClientName",
            "ExchangeName": "MyExchangeName"
        }
    }
}
```

```

### ### The Options Classes

``AbpRabbitMqOptions`` and ``AbpRabbitMqEventBusOptions`` classes can be used to configure the connection strings and event bus options for the RabbitMQ.

You can configure this options inside the `ConfigureServices` of your [module](Module-Development-Basics.md).

**\*\*Example: Configure the connection\*\***

```

```csharp
Configure<AbpRabbitMqOptions>(options =>
{
    options.Connections.Default.UserName = "user";
    options.Connections.Default.Password = "pass";
    options.Connections.Default.HostName = "123.123.123.123";
    options.Connections.Default.Port = 5672;
});
```

```

**\*\*Example: Configure the client, exchange names and prefetchCount\*\***

```

```csharp
Configure<AbpRabbitMqEventBusOptions>(options =>
{
    options.ClientName = "TestApp1";
    options.ExchangeName = "TestMessages";
    options.PrefetchCount = 1;
});
```

```

Using these options classes can be combined with the `appsettings.json` way. Configuring an option property in the code overrides the value in the configuration file.

#### 7.13.3.3 Kafka Integration

##### # Distributed Event Bus Kafka Integration

> This document explains \*\*how to configure the [Kafka](https://kafka.apache.org/)\*\* as the distributed event bus provider. See the [distributed event bus document](Distributed-Event-Bus.md) to learn how to use the distributed event bus system

##### ## Installation

Use the ABP CLI to add [\[Volo.Abp.EventBus.Kafka\]](https://www.nuget.org/packages/Volo.Abp.EventBus.Kafka)(<https://www.nuget.org/packages/Volo.Abp.EventBus.Kafka>) NuGet package to your project:

- \* Install the [\[ABP CLI\]](https://docs.abp.io/en/abp/latest/CLI)(<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.EventBus.Kafka` package.
- \* Run `abp add-package Volo.Abp.EventBus.Kafka` command.

If you want to do it manually, install the [\[Volo.Abp.EventBus.Kafka\]](https://www.nuget.org/packages/Volo.Abp.EventBus.Kafka)(<https://www.nuget.org/packages/Volo.Abp.EventBus.Kafka>) NuGet package to your project and add `'[DependsOn(typeof(AbpEventBusKafkaModule))]'` to the [\[ABP module\]](#)(Module-Development-Basics.md) class inside your project.

## ## Configuration

You can configure using the standard [\[configuration system\]](#)(Configuration.md), like using the `appsettings.json` file, or using the [\[options\]](#)(Options.md) classes.

### ### `appsettings.json` file configuration

This is the simplest way to configure the Kafka settings. It is also very strong since you can use any other configuration source (like environment variables) that is [supported by the AspNet Core](#)(<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/configuration/>).

**\*\*Example: The minimal configuration to connect to a local kafka server with default configurations\*\***

```
```json
{
  "Kafka": {
    "Connections": {
      "Default": {
        "BootstrapServers": "localhost:9092"
      }
    },
    "EventBus": {
      "GroupId": "MyGroupId",
      "TopicName": "MyTopicName"
    }
  }
}...
```

* `MyGroupId` is the name of this application, which is used as the **GroupId** on the Kafka.
* `MyTopicName` is the **topic name**.

See [\[the Kafka document\]](https://docs.confluent.io/current/clients/confluent-kafka-dotnet/api/Confluent.Kafka.html)(<https://docs.confluent.io/current/clients/confluent-kafka-dotnet/api/Confluent.Kafka.html>) to understand these options better.

Connections

If you need to connect to another server than the localhost, you need to configure the connection properties.

****Example: Specify the host name (as an IP address)****

```
```json
{
 "Kafka": {
 "Connections": {
 "Default": {
 "BootstrapServers": "123.123.123.123:9092"
 }
 },
 "EventBus": {
 "GroupId": "MyGroupId",
 "TopicName": "MyTopicName"
 }
 }
}...```

```

Defining multiple connections is allowed. In this case, you can specify the connection that is used for the event bus.

**\*\*Example: Declare two connections and use one of them for the event bus\*\***

```
```json
{
  "Kafka": {
    "Connections": {
      "Default": {
        "BootstrapServers": "123.123.123.123:9092"
      },
      "SecondConnection": {
        "BootstrapServers": "321.321.321.321:9092"
      }
    },
    "EventBus": {
      "GroupId": "MyGroupId",
      "TopicName": "MyTopicName",
      "ConnectionName": "SecondConnection"
    }
  }
}...```

```

This allows you to use multiple Kafka cluster in your application, but select one of them for the event bus.

You can use any of the [ClientConfig](<https://docs.confluent.io/current/clients/confluent-kafka-dotnet/api/Confluent.Kafka.ClientConfig.html>) properties as the connection properties.

****Example: Specify the socket timeout****

```
```json
{```

```

```
"Kafka": {
 "Connections": {
 "Default": {
 "BootstrapServers": "123.123.123.123:9092",
 "SocketTimeoutMs": 60000
 }
 }
}...
}
```

### ### The Options Classes

`AbpKafkaOptions` and `AbpKafkaEventBusOptions` classes can be used to configure the connection strings and event bus options for the Kafka.

You can configure this options inside the `ConfigureServices` of your [module](Module-Development-Basics.md).

#### \*\*Example: Configure the connection\*\*

```
```csharp
Configure<AbpKafkaOptions>(options =>
{
    options.Connections.Default.BootstrapServers = "123.123.123.123:9092";
    options.Connections.Default.SaslUsername = "user";
    options.Connections.Default.SaslPassword = "pwd";
});  
...  
```

```

#### \*\*Example: Configure the consumer config\*\*

```
```csharp
Configure<AbpKafkaOptions>(options =>
{
    options.ConfigureConsumer = config =>
    {
        config.GroupId = "MyGroupId";
        config.EnableAutoCommit = false;
    };
});  
...  
```

```

#### \*\*Example: Configure the producer config\*\*

```
```csharp
Configure<AbpKafkaOptions>(options =>
{
    options.ConfigureProducer = config =>
    {
        config.MessageTimeoutMs = 6000;
        config.Acks = Acks.All;
    };
});  
...  
```

```

#### \*\*Example: Configure the topic specification\*\*

```
```csharp
Configure<AbpKafkaOptions>(options =>
{
    options.ConfigureTopic = specification =>
    {
        specification.ReplicationFactor = 3;
        specification.NumPartitions = 3;
    };
});
```

```

Using these options classes can be combined with the `'appsettings.json'` way. Configuring an option property in the code overrides the value in the configuration file.

#### 7.13.3.4 Rebus Integration

##### # Distributed Event Bus Rebus Integration

> This document explains \*\*how to configure the [\[Rebus\]](#)(<http://mookid.dk/category/rebus/>)\*\* as the distributed event bus provider. See the [\[distributed event bus document\]](#)(Distributed-Event-Bus.md) to learn how to use the distributed event bus system

##### ## Installation

Use the ABP CLI to add [\[Volo.Abp.EventBus.Rebus\]](#)(<https://www.nuget.org/packages/Volo.Abp.EventBus.Rebus>) NuGet package to your project:

- \* Install the [\[ABP CLI\]](#)(<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- \* Open a command line (terminal) in the directory of the `'.csproj'` file you want to add the `'Volo.Abp.EventBus.Rebus'` package.
- \* Run `'abp add-package Volo.Abp.EventBus.Rebus'` command.

If you want to do it manually, install the [\[Volo.Abp.EventBus.Rebus\]](#)(<https://www.nuget.org/packages/Volo.Abp.EventBus.Rebus>) NuGet package to your project and add `'[DependsOn(typeof(AbpEventBusRebusModule))]'` to the [\[ABP module\]](#)(Module-Development-Basics.md) class inside your project.

##### ## Configuration

You can configure using the standard [\[configuration system\]](#)(Configuration.md), like using the [\[options\]](#)(Options.md) classes.

##### ### The Options Classes

`'AbpRebusEventBusOptions'` class can be used to configure the event bus options for the Rebus.

You can configure this options inside the `'PreConfigureServices'` of your [\[module\]](#)(Module-Development-Basics.md).

\*\*Example: Minimize configuration\*\*

```
```csharp
PreConfigure<AbpRebusEventBusOptions>(options =>
{
    options.InputQueueName = "eventbus";
});
```

Rebus has many options, you can use the `Configurer` property of `AbpRebusEventBusOptions` class to configure.

Default events are **stored in memory**. See the [rebus document](<https://github.com/rebus-org/Rebus/wiki/Transport>) for more details.

Example: Configure the store

```
```csharp
PreConfigure<AbpRebusEventBusOptions>(options =>
{
 options.InputQueueName = "eventbus";
 options.Configurer = rebusConfigurer =>
 {
 rebusConfigurer.Transport(t => t.UseMsmq("eventbus"));
 rebusConfigurer.Subscriptions(s =>
s.UseJsonFile(@"subscriptions.json"));
 };
});
```

You can use the `Publish` property of `AbpRebusEventBusOptions` class to change the publishing method

**Example: Configure the event publishing**

```
```csharp
PreConfigure<AbpRebusEventBusOptions>(options =>
{
    options.InputQueueName = "eventbus";
    options.Publish = async (bus, type, data) =>
    {
        await bus.Publish(data);
    };
});
```

7.14 Features

Features

ABP Feature system is used to **enable**, **disable** or **change the behavior** of the application features **on runtime**.

The runtime value for a feature is generally a `boolean` value, like `true` (enabled) or `false` (disabled). However, you can get/set **any kind** of value for feature.

Feature system was originally designed to control the tenant features in a **[multi-tenant](Multi-Tenancy.md)** application. However, it is **extensible** and capable of determining the features by any condition.

> The feature system is implemented with the [Volo.Abp.Features](https://www.nuget.org/packages/Volo.Abp.Features) NuGet package. Most of the times you don't need to manually [install it](https://app.io/package-detail/Volo.Abp.Features) since it comes pre-installed with the [application startup template](Startup-Templates/Application.md).

Checking for the Features

Before explaining to define features, let's see how to check a feature value in your application code.

RequiresFeature Attribute

`[RequiresFeature]` attribute (defined in the `Volo.Abp.Features` namespace) is used to declaratively check if a feature is `true` (enabled) or not. It is a useful shortcut for the `boolean` features.

Example: Check if the "PDF Reporting" feature enabled

```
```csharp
public class ReportingAppService : ApplicationService, IReportingAppService
{
 [RequiresFeature("MyApp.PdfReporting")]
 public async Task<PdfReportResultDto> GetPdfReportAsync()
 {
 //TODO...
 }
}..
```

- \* `RequiresFeature(...)` simply gets a feature name to check if it is enabled or not. If not enabled, an authorization [exception](Exception-Handling.md) is thrown and a proper response is returned to the client side.
- \* `[RequiresFeature]` can be used for a \*\*method\*\* or a \*\*class\*\*. When you use it for a class, all the methods of that class require the given feature.
- \* `RequiresFeature` may get multiple feature names, like `RequiresFeature("Feature1", "Feature2")`. In this case ABP checks if any of the features enabled. Use `RequiresAll` option, like `RequiresFeature("Feature1", "Feature2", RequiresAll = true)` to force to check all of the features to be enabled.
- \* Multiple usage of `RequiresFeature` attribute is supported for a method or class. ABP checks all of them in that case.

> Feature name can be any arbitrary string. It should be unique for a feature.

### #### About the Interception

ABP Framework uses the interception system to make the `RequiresFeature` attribute working. So, it can work with any class (application services, controllers...) that is injected from the [dependency injection](Dependency-Injection.md).

However, there are **some rules should be followed** in order to make it working;

\* If you are **not injecting** the service over an interface (like `'IMyService'`), then the methods of the service must be `'virtual'`. Otherwise, [dynamic proxy / interception](Dynamic-Proxying-Interceptors.md) system can not work.

\* Only `'async'` methods (methods returning a `'Task'` or `'Task<T>'`) are intercepted.

> There is an exception for the **controller and razor page methods**. They **don't require** the following the rules above, since ABP Framework uses the action/page filters to implement the feature checking in this case.

### ### IFeatureChecker Service

`'IFeatureChecker'` allows to check a feature in your application code.

#### #### IsEnabledAsync

Returns `'true'` if the given feature is enabled. So, you can conditionally execute your business flow.

**Example:** Check if the "PDF Reporting" feature enabled\*

```
```csharp
public class ReportingAppService : ApplicationService, IReportingAppService
{
    private readonly IFeatureChecker _featureChecker;

    public ReportingAppService(IFeatureChecker featureChecker)
    {
        _featureChecker = featureChecker;
    }

    public async Task<PdfReportResultDto> GetPdfReportAsync()
    {
        if (await _featureChecker.IsEnabledAsync("MyApp.PdfReporting"))
        {
            //TODO...
        }
        else
        {
            //TODO...
        }
    }
}..
```

`'IsEnabledAsync'` has overloads to check multiple features in one method call.

GetOrNullAsync

Gets the current value for a feature. This method returns a `'string'`, so you store any kind of value inside it, by converting to or from `'string'`.

Example: Check the maximum product count allowed*

```

```csharp
public class ProductController : AbpController
{
 private readonly IFeatureChecker _featureChecker;

 public ProductController(IFeatureChecker featureChecker)
 {
 _featureChecker = featureChecker;
 }

 public async Task<IActionResult> Create(CreateProductModel model)
 {
 var currentProductCount = await GetCurrentProductCountFromDatabase();

 //GET THE FEATURE VALUE
 var maxProductCountLimit =
 await _featureChecker.GetOrNullAsync("MyApp.MaxProductCount");

 if (currentProductCount >= Convert.ToInt32(maxProductCountLimit))
 {
 throw new BusinessException(
 "MyApp:ReachToMaxProductCountLimit",
 $"You can not create more than {maxProductCountLimit}");
 }
 }

 //TODO: Create the product in the database...
}

private async Task<int> GetCurrentProductCountFromDatabase()
{
 throw new NotImplementedException();
}
```

```

This example uses a numeric value as a feature limit product counts for a user/tenant in a SaaS application.

Instead of manually converting the value to `int`, you can use the generic overload of the `GetAsync` method:

```

```csharp
var maxProductCountLimit = await
 _featureChecker.GetAsync<int>("MyApp.MaxProductCount");
```

```

Extension Methods

There are some useful extension methods for the `IFeatureChecker` interface;

- * `Task<T> GetAsync<T>(string name, T defaultValue = default)": Used to get a value of a feature with the given type `T`. Allows to specify a `defaultValue` that is returned when the feature value is `null`.

- * `CheckEnabledAsync(string name)`: Checks if given feature is enabled. Throws an `AbpAuthorizationException` if the feature was not `true` (enabled).

Defining the Features

A feature should be defined to be able to check it.

FeatureDefinitionProvider

Create a class inheriting the `FeatureDefinitionProvider` to define features.

****Example: Defining features****

```
```csharp
using Volo.Abp.Features;

namespace FeaturesDemo
{
 public class MyFeatureDefinitionProvider : FeatureDefinitionProvider
 {
 public override void Define(IFeatureDefinitionContext context)
 {
 var myGroup = context.AddGroup("MyApp");

 myGroup.AddFeature("MyApp.PdfReporting", defaultValue: "false");
 myGroup.AddFeature("MyApp.MaxProductCount", defaultValue: "10");
 }
 }
}..
```

> ABP automatically discovers this class and registers the features. No additional configuration required.

> This class is generally created in the `Application.Contracts` project of your solution.

- \* In the `Define` method, you first need to add a \*\*feature group\*\* for your application/module or get an existing group then add \*\*features\*\* to this group.
- \* First feature, named `MyApp.PdfReporting`, is a `boolean` feature with `false` as the default value.
- \* Second feature, named `MyApp.MaxProductCount`, is a numeric feature with `10` as the default value.

Default value is used if there is no other value set for the current user/tenant.

**### Other Feature Properties**

While these minimal definitions are enough to make the feature system working, you can specify the \*\*optional properties\*\* for the features;

- \* `DisplayName`: A localizable string that will be used to show the feature name on the user interface.
- \* `Description`: A longer localizable text to describe the feature.

- \* `ValueType`: Type of the feature value. Can be a class implementing the `IStringValue` type. Built-in types:
  - \* `ToggleStringValueType`: Used to define `true`/`false`, `on`/`off`, `enabled`/`disabled` style features. A checkbox is shown on the UI.
  - \* `FreeTextStringValueType`: Used to define free text values. A textbox is shown on the UI.
  - \* `SelectionStringValueType`: Used to force the value to be selected from a list. A dropdown list is shown on the UI.
- \* `IsVisibleToClients` (default: `true`): Set false to hide the value of this feature from clients (browsers). Sharing the value with the clients helps them to conditionally show/hide/change the UI parts based on the feature value.
- \* `Properties`: A dictionary to set/get arbitrary key-value pairs related to this feature. This can be a point for customization.

So, based on these descriptions, it would be better to define these features as shown below:

```
```csharp
using FeaturesDemo.Localization;
using Volo.Abp.Features;
using Volo.Abp.Localization;
using Volo.Abp.Validation.StringValues;

namespace FeaturesDemo
{
    public class MyFeatureDefinitionProvider : FeatureDefinitionProvider
    {
        public override void Define(IFeatureDefinitionContext context)
        {
            var myGroup = context.AddGroup("MyApp");

            myGroup.AddFeature(
                "MyApp.PdfReporting",
                defaultValue: "false",
                displayName: LocalizableString
                    .Create<FeaturesDemoResource>("PdfReporting")
),
                valueType: new ToggleStringValueType()
            );

            myGroup.AddFeature(
                "MyApp.MaxProductCount",
                defaultValue: "10",
                displayName: LocalizableString
                    .Create<FeaturesDemoResource>("MaxProductCou
nt"),
                valueType: new FreeTextStringValueType(
                    new NumericValueValidator(0, 1000000))
            );
        }
    }
}

* `FeaturesDemoResource` is the project name in this example code. See the [localization document](Localization.md) for details about the localization system.
```

```
* First feature is set to `ToggleStringValue`, while the second one is  
set to `FreeTextStringValue` with a numeric validator that allows to the  
values from `0` to `1,000,000`.
```

Remember to define the localization the keys in your localization file:

```
```json  
"PdfReporting": "PDF Reporting",
"MaxProductCount": "Maximum number of products"
```
```

See the [\[localization document\]](#)(Localization.md) for details about the localization system.

Feature Management Modal

The [\[application startup template\]](#)(Startup-Templates/Application.md) comes with the [\[tenant management\]](#)(Modules/Tenant-Management.md) and the [\[feature management\]](#)(Modules/Feature-Management.md) modules pre-installed.

Whenever you define a new feature, it will be available on the **feature management modal**. To open this modal, navigate to the **tenant management page** and select the `Features` action for a tenant (create a new tenant if there is no tenant yet):

```
![features-action](images/features-action.png)
```

This action opens a modal to manage the feature values for the selected tenant:

```
![features-modal](images/features-modal.png)
```

So, you can enable, disable and set values for a tenant. These values will be used whenever a user of this tenant uses the application.

See the **Feature Management** section below to learn more about managing the features.

Child Features

A feature may have child features. This is especially useful if you want to create a feature that is selectable only if another feature was enabled.

****Example: Defining child features****

```
```csharp  
using FeaturesDemo.Localization;
using Volo.Abp.Features;
using Volo.Abp.Localization;
using Volo.Abp.Validation.StringValues;

namespace FeaturesDemo
{
 public class MyFeatureDefinitionProvider : FeatureDefinitionProvider
 {
 public override void Define(IFeatureDefinitionContext context)
 {
 var myGroup = context.AddGroup("MyApp");
```

```

 var reportingFeature = myGroup.AddFeature(
 "MyApp.Reporting",
 defaultValue: "false",
 displayName: LocalizableString
 .Create<FeaturesDemoResource>("Reporting"),
 valueType: new ToggleStringValueType()
);

 reportingFeature.CreateChild(
 "MyApp.PdfReporting",
 defaultValue: "false",
 displayName: LocalizableString
 .Create<FeaturesDemoResource>("PdfReporting")
 ,
 valueType: new ToggleStringValueType()
);

 reportingFeature.CreateChild(
 "MyApp.ExcelReporting",
 defaultValue: "false",
 displayName: LocalizableString
 .Create<FeaturesDemoResource>("ExcelReportin
g"),
 valueType: new ToggleStringValueType()
);
 }
}
```

```

The example above defines a **Reporting** feature with two children: **PDF Reporting** and **Excel Reporting**.

Changing Features Definitions of a Depended Module

A class deriving from the `'FeatureDefinitionProvider'` (just like the example above) can also get the existing feature definitions (defined by the depended [modules](Module-Development-Basics.md)) and change their definitions.

****Example: Manipulate an existing feature definition****

```

```csharp
var someGroup = context.GetGroupOrNull("SomeModule");
var feature = someGroup.Features.FirstOrDefault(f => f.Name ==
"SomeFeature");
if (feature != null)
{
 feature.Description = ...
 feature.CreateChild(...);
}
```

```

Check a Feature in the Client Side

A feature value is available at the client side too, unless you set `'IsVisibleToClients'` to `'false'` on the feature definition. The feature values

are exposed from the [Application Configuration API](API/Application-Configuration.md) and usable via some services on the UI.

See the following documents to learn how to check features in different UI types:

- * [ASP.NET Core MVC / Razor Pages / JavaScript API](UI/AspNetCore/JavaScript-API/Features.md)
- * [Angular](UI/Angular/Features.md)

Blazor applications can use the same `IFeatureChecker` service as explained above.

Feature Management

Feature management is normally done by an admin user using the feature management modal:

![features-modal](images/features-modal.png)

This modal is available on the related entities, like tenants in a multi-tenant application. To open it, navigate to the **Tenant Management** page (for a multi-tenant application), click to the **Actions** button left to the Tenant and select the **Features** action.

If you need to manage features by code, inject the `IFeatureManager` service.

Example: Enable PDF reporting for a tenant

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IFeatureManager _featureManager;

 public MyService(IFeatureManager featureManager)
 {
 _featureManager = featureManager;
 }

 public async Task EnablePdfReporting(Guid tenantId)
 {
 await _featureManager.SetForTenantAsync(
 tenantId,
 "MyApp.PdfReporting",
 true.ToString()
);
 }
}..
```

`IFeatureManager` is defined by the Feature Management module. It comes pre-installed with the application startup template. See the [feature management module documentation](Modules/Feature-Management.md) for more information.

## ## Advanced Topics

### ### Feature Value Providers

Feature system is extensible. Any class derived from `FeatureValueProvider` (or implements `IFeatureValueProvider`) can contribute to the feature system. A value provider is responsible to \*\*obtain the current value\*\* of a given feature.

Feature value providers are \*\*executed one by one\*\*. If one of them return a non-null value, then this feature value is used and the other providers are not executed.

There are three pre-defined value providers, executed by the given order:

- \* `TenantFeatureValueProvider` tries to get if the feature value is explicitly set for the \*\*current tenant\*\*.
- \* `EditionFeatureValueProvider` tries to get the feature value for the current edition. Edition Id is obtained from the current principal identity (`ICurrentPrincipalAccessor`) with the claim name `editionid` (a constant defined as `AbpClaimTypes.EditionId`). Editions are not implemented for the [tenant management](Modules/Tenant-Management.md) module. You can implement it yourself or consider to use the [SaaS module](https://commercial.abp.io/modules/Volo.Saas) of the ABP Commercial.
- \* `DefaultValueFeatureValueProvider` gets the default value of the feature.

You can write your own provider by inheriting the `FeatureValueProvider`.

**Example: Enable all features for a user with "SystemAdmin" as a "User\_Type" claim value**

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.Features;
using Volo.Abp.Security.Claims;
using Volo.Abp.Validation.StringValues;

namespace FeaturesDemo
{
    public class SystemAdminFeatureValueProvider : FeatureValueProvider
    {
        public override string Name => "SA";

        private readonly ICurrentPrincipalAccessor _currentPrincipalAccessor;

        public SystemAdminFeatureValueProvider(
            IFeatureStore featureStore,
            ICurrentPrincipalAccessor currentPrincipalAccessor)
            : base(featureStore)
        {
            _currentPrincipalAccessor = currentPrincipalAccessor;
        }

        public override Task<string> GetOrNullAsync(FeatureDefinition feature)
        {
            if (feature.ValueType is ToggleStringValue &&
                _currentPrincipalAccessor.Principal?.FindFirst("User_Type")?.Value == "SystemAdmin")
            {
                return Task.FromResult("true");
            }
        }
    }
}
```

```

        }
        return null;
    }
}
```

```

If a provider returns `null`, then the next provider is executed.

Once a provider is defined, it should be added to the `AbpFeatureOptions` as shown below:

```

```csharp
Configure<AbpFeatureOptions>(options =>
{
    options.ValueProviders.Add<SystemAdminFeatureValueProvider>();
});
```

```

Use this code inside the `ConfigureServices` of your [module](Module-Basics.md) class.

#### ### Feature Store

`IFeatureStore` is the only interface that needs to be implemented to read the value of features from a persistence source, generally a database system. The Feature Management module implements it and pre-installed in the application startup template. See the [feature management module documentation](<https://docs.abp.io/en/abp/latest/Modules/Feature-Management>) for more information

## 7.15 Global Features

#### # Global Features

Global Feature system is used to enable/disable an application feature on development time. It is done on the development time, because some \*\*services\*\* (e.g. controllers) are removed from the application model and \*\*database tables\*\* are not created for the disabled features, which is not possible on runtime.

Global Features system is especially useful if you want to develop a reusable application module with optional features. If the final application doesn't want to use some of the features, it can disable these features.

> If you are looking for a system to enable/disable features based on current tenant or any other condition, please see the [Features](Features.md) document.

#### ## Installation

> This package is already installed by default with the startup template. So, most of the time, you don't need to install it manually.

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.GlobalFeatures
```

```

### ## Defining a Global Feature

A feature class is something like that:

```
```csharp
[GlobalFeatureName("Shopping.Payment")]
public class PaymentFeature
{
}
```

```

### ## Enable/Disable Global Features

Use `'GlobalFeatureManager.Instance'` to enable/disable a global feature.

```
```csharp
// Able to Enable/Disable with generic type parameter.
GlobalFeatureManager.Instance.Enable<PaymentFeature>();
GlobalFeatureManager.Instance.Disable<PaymentFeature>();

// Also able to Enable/Disable with string feature name.
GlobalFeatureManager.Instance.Enable("Shopping.Payment");
GlobalFeatureManager.Instance.Disable("Shopping.Payment");
```

```

> Global Features are disabled unless they are explicitly enabled.

### ### Where to Configure Global Features?

Global Features have to be configured before application startup. Since the `'GlobalFeatureManager.Instance'` is a singleton object, one-time, static configuration is enough. It is suggested to enable/disable global features in `'PreConfigureServices'` method of your module. You can use the `'OneTimeRunner'` utility class to make sure it runs only once:

```
```csharp
private static readonly OneTimeRunner OneTimeRunner = new OneTimeRunner();
public override void PreConfigureServices(ServiceConfigurationContext context)
{
    OneTimeRunner.Run(() =>
    {
        GlobalFeatureManager.Instance.Enable<PaymentFeature>();
    });
}
```

```

### ## Check for a Global Feature

```
```csharp
GlobalFeatureManager.Instance.IsEnabled<PaymentFeature>()
GlobalFeatureManager.Instance.IsEnabled("Shopping.Payment")
```

```

Both methods return `bool`. So, you can write conditional logic as shown below:

```
```csharp
if (GlobalFeatureManager.Instance.IsEnabled<PaymentFeature>())
{
    // Some strong payment codes here...
}..
```

RequiresGlobalFeature Attribute

Beside the manual check, there is `[RequiresGlobalFeature]` attribute to check it declaratively for a controller or page. ABP returns HTTP Response `404` if the related feature was disabled.

```
```csharp
[RequiresGlobalFeature(typeof(PaymentFeature))]
public class PaymentController : AbpController
{
}
```

## Grouping Features of a Module

It is common to group global features of a module to allow the final application developer easily discover and configure the features. Following example shows how to group features of a module.

Assume that we've defined a global feature for `Subscription` feature of an `Ecommerce` module:

```
```csharp
[GlobalFeatureName("Ecommerce.Subscription")]
public class SubscriptionFeature : GlobalFeature
{
    public SubscriptionFeature(GlobalModuleFeatures module)
        : base(module)
    {
    }
}..
```

You can define as many features as you need in your module. Then define a class to group these features together:

```
```csharp
public class GlobalEcommerceFeatures : GlobalModuleFeatures
{
 public const string ModuleName = "Ecommerce";

 public SubscriptionFeature Subscription =>
GetFeature<SubscriptionFeature>();

 public GlobalEcommerceFeatures(GlobalFeatureManager featureManager)
 : base(featureManager)
```

```

 {
 AddFeature(new SubscriptionFeature(this));
 }
}..

```

Finally, you can create an extension method on `'GlobalModuleFeaturesDictionary'`:

```

```csharp
public static class GlobalModuleFeaturesDictionaryEcommerceExtensions
{
    public static GlobalEcommerceFeatures Ecommerce(
        this GlobalModuleFeaturesDictionary modules)
    {
        return modules.GetOrAdd(
            GlobalEcommerceFeatures.ModuleName,
            _ => new GlobalEcommerceFeatures(modules.FeatureManager)
        ) as GlobalEcommerceFeatures;
    }
}..

```

Then `'GlobalFeatureManager.Instance.Modules.Ecommerce()'` can be used to access the global features of your module. Examples usages:

```

```csharp
GlobalFeatureManager.Instance.Modules.Ecommerce().Subscription.Enable();
GlobalFeatureManager.Instance.Modules.Ecommerce().EnableAll();
```

```

7.16 GUID Generation

GUID Generation

GUID is a common **primary key type** that is used in database management systems. ABP Framework prefers GUID as the primary for pre-built [application modules](Modules/Index.md). Also, `'ICurrentUser.Id'` property ([see](CurrentUser.md)) is type of GUID, that means the ABP Framework assumes that the User Id is always GUID.

Why Prefer GUID?

GUID has advantages and disadvantages. You can find many articles on the web related to this topic, so we will not discuss all again, but will list the most fundamental advantages:

- * It is **usable** in all database providers.
- * It allows to **determine the primary key** on the client side, without needing a **database round trip** to generate the Id value. This can be more performant while inserting new records to the database and allows us to know the PK before interacting to the database.
- * GUIDs are **naturally unique** which has some advantages in the following situations if;
 - * You need to integrate to **external** systems.
 - * You need to **split or merge** different tables.

```
* You are creating **distributed systems**.  
* GUIDs are impossible to guess, so they can be **more secure** compared to  
auto-increment Id values in some cases.
```

While there are some disadvantages (just search it on the web), we found these advantages much more important while designing the ABP Framework.

IGuidGenerator

The most important problem with GUID is that it is **not sequential by default**. When you use the GUID as the primary key and set it as the **clustered index** (which is default) for your table, it brings a significant **performance problem on insert** (because inserting new record may need to re-order the existing records).

So, **never use `Guid.NewGuid()` to create Ids** for your entities!

One good solution to this problem is to generate **sequential GUIDs**, which is provided by the ABP Framework out of the box. `IGuidGenerator` service creates sequential GUIDs (implemented by the `SequentialGuidGenerator` by default). Use `IGuidGenerator.Create()` when you need to manually set Id of an [entity](Entities.md).

Example: An entity with GUID primary key and creating the entity

Assume that you've a `Product` [entity](Entities.md) that has a `Guid` key:

```
```csharp  
using System;
using Volo.Abp.Domain.Entities;

namespace AbpDemo
{
 public class Product : AggregateRoot<Guid>
 {
 public string Name { get; set; }

 private Product() /* This constructor is used by the ORM/database provider */

 public Product(Guid id, string name)
 : base(id)
 {
 Name = name;
 }
 }
}...````
```

And you want to create a new product:

```
```csharp  
using System;  
using System.Threading.Tasks;  
using Volo.Abp.DependencyInjection;  
using Volo.Abp.Domain.Repositories;  
using Volo.Abp.Guids;
```

```

namespace AbpDemo
{
    public class MyProductService : ITransientDependency
    {
        private readonly IRepository<Product, Guid> _productRepository;
        private readonly IGuidGenerator _guidGenerator;

        public MyProductService(
            IRepository<Product, Guid> productRepository,
            IGuidGenerator guidGenerator)
        {
            _productRepository = productRepository;
            _guidGenerator = guidGenerator;
        }

        public async Task CreateAsync(string productName)
        {
            var product = new Product(_guidGenerator.Create(), productName);

            await _productRepository.InsertAsync(product);
        }
    }
}
...

```

This service injects the `IGuidGenerator` in the constructor. If your class is an [application service](Application-Services.md) or deriving from one of the other base classes, you can directly use the `GuidGenerator` base property which is a pre-injected `IGuidGenerator` instance.

Options

AbpSequentialGuidGeneratorOptions

`AbpSequentialGuidGeneratorOptions` is the [option class](Options.md) that is used to configure the sequential GUID generation. It has a single property:

- * `DefaultSequentialGuidType` (`enum` of type `SequentialGuidType`): The strategy used while generating GUID values.

Database providers behaves differently while processing GUIDs, so you should set it based on your database provider. `SequentialGuidType` has the following `enum` members:

- * `SequentialAtEnd` (**default**) works well with the [SQL Server](Entity-Framework-Core.md).
- * `SequentialAsString` is used by [MySQL](Entity-Framework-Core-MySQL.md) and [PostgreSQL](Entity-Framework-Core-PostgreSQL.md).
- * `SequentialAsBinary` is used by [Oracle](Entity-Framework-Core-Oracle.md).

Configure this option in the `ConfigureServices` method of your [module](Module-Development-Basics.md), as shown below:

```

```csharp
Configure<AbpSequentialGuidGeneratorOptions>(options =>
{
 options.DefaultSequentialGuidType =
 SequentialGuidType.SequentialAsBinary;
}
```

```

```
});
```

> EF Core [[integration packages](https://docs.abp.io/en/abp/latest/Entity-Framework-Core-Other-DBMS)](<https://docs.abp.io/en/abp/latest/Entity-Framework-Core-Other-DBMS>) sets this option to a proper value for the related DBMS. So, most of the times, you don't need to set this option if you are using these integration packages.

7.17 Image Manipulation

```
# Image Manipulation
ABP Framework provides services to compress and resize images and implements these services with popular [ImageSharp](https://sixlabors.com/products/imagesharp/) and [Magick.NET](https://github.com/dlemstra/Magick.NET) libraries. You can use these services in your reusable modules, libraries and applications, so you don't depend on a specific imaging library.
```

> The image resizer/compressor system is designed to be extensible. You can implement your own image resizer/compressor contributor and use it in your application.

Installation

You can add this package to your application by either using the [[ABP CLI](#)](CLI.md) or manually installing it. Using the [[ABP CLI](#)](CLI.md) is the recommended approach.

Using the ABP CLI

Open a command line terminal in the folder of your project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Imaging.Abstractions
````
```

Manual Installation

If you want to manually install;

1. Add the [[Volo.Abp.Imaging.Abstractions](https://www.nuget.org/packages/Volo.Abp.Imaging.Abstractions)](<https://www.nuget.org/packages/Volo.Abp.Imaging.Abstractions>) NuGet package to your project:

```
```
Install-Package Volo.Abp.Imaging.Abstractions
````
```

2. Add the `AbpImagingAbstractionsModule` to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpImagingAbstractionsModule) //Add the new module dependency
)]
```

```

public class YourModule : AbpModule
{
}
```
## Providers

ABP Framework provides two image resizer/compressor implementations out of the box:

* [Magick.NET](#magick-net-provider)
* [ImageSharp](#imagesharp-provider)

You should install one of these providers to make it actually working.

> If none of the provider packages installed into your application, compress/resize operations return the untouched input image.

## IImageResizer

You can [inject](Dependency-Injection.md) the `IImageResizer` service and use it for image resize operations. Here is the available methods of the `IImageResizer` service:

```csharp
public interface IImageResizer
{
 /* Works with a Stream object that represents an image */
 Task<ImageResizeResult<Stream>> ResizeAsync(
 Stream stream,
 ImageResizeArgs resizeArgs,
 string mimeType = null,
 CancellationToken cancellationToken = default
);

 /* Works with a byte array that contains an image file */
 Task<ImageResizeResult<byte[]>> ResizeAsync(
 byte[] bytes,
 ImageResizeArgs resizeArgs,
 string mimeType = null,
 CancellationToken cancellationToken = default
);
}
```

**Example usage:**


```csharp
var result = await _imageResizer.ResizeAsync(
 stream, /* A stream object that represents an image */
 new ImageResizeArgs
 {
 Width = 100,
 Height = 100,
 Mode = ImageResizeMode.Crop
 },
 mimeType: "image/jpeg"
);

```

```
...
```

> You can use `MimeTypes.Image.Jpeg` constant instead of the `image/jpeg` magic string used in that example.

### ### ImageResizeArgs

The `ImageResizeArgs` is a class that is used to define the resize operation parameters. It has the following properties:

- \* `Width`: The width of the resized image.
- \* `Height`: The height of the resized image.
- \* `Mode`: The resize mode (see the [ImageResizeMode](#imageresizemode) section for more information).

### ### ImageResizeMode

The `ImageResizeMode` is an enum that is used to define the resize mode. It has the following values:

```
```csharp
public enum ImageResizeMode : byte
{
    None = 0,
    Stretch = 1,
    BoxPad = 2,
    Min = 3,
    Max = 4,
    Crop = 5,
    Pad = 6,
    Default = 7
}
````
```

> See the [ImageSharp documentation](<https://docs.sixlabors.com/api/ImageSharp/SixLabors.ImageSharp.Processing.ResizeMode.html>) for more information about the resize modes.

### ### ImageResizeResult

The `ImageResizeResult` is a generic class that is used to return the result of the image resize operations. It has the following properties:

- \* `Result`: The resized image (stream or byte array).
- \* `State`: The result of the resize operation (type: `ImageProcessState`).

### ### ImageProcessState

The `ImageProcessState` is an enum that is used to return the the result of the image resize operations. It has the following values:

```
```csharp
public enum ImageProcessState : byte
{
    Done = 1,
    Canceled = 2,
    Unsupported = 3,
}
```

```
...
```

ImageResizeOptions

`ImageResizeOptions` is an [options object](Options.md) that is used to configure the image resize system. It has the following properties:

- * `DefaultResizeMode`: The default resize mode. (Default: `ImageResizeMode.None`)

IImageCompressor

You can [inject](Dependency-Injection.md) the `IImageCompressor` service and use it for image compression operations. Here is the available methods of the `IImageCompressor` service:

```
```csharp
public interface IImageCompressor
{
 /* Works with a Stream object that represents an image */
 Task<ImageCompressResult<Stream>> CompressAsync(
 Stream stream,
 string mimeType = null,
 CancellationToken cancellationToken = default
);

 /* Works with a byte array that contains an image file */
 Task<ImageCompressResult<byte[]>> CompressAsync(
 byte[] bytes,
 string mimeType = null,
 CancellationToken cancellationToken = default
);
}
```

\*\*Example usage:\*\*

```
```csharp
var result = await _imageCompressor.CompressAsync(
    stream, /* A stream object that represents an image */
    mimeType: "image/jpeg"
);
```

ImageCompressResult

The `ImageCompressResult` is a generic class that is used to return the result of the image compression operations. It has the following properties:

- * `Result`: The compressed image (stream or byte array).
- * `State`: The result of the compress operation (type: `ImageProcessState`).

ImageProcessState

The `ImageProcessState` is an enum that is used to return the the result of the image compress operations. It has the following values:

```
```csharp
```

```
public enum ImageProcessState : byte
{
 Done = 1,
 Canceled = 2,
 Unsupported = 3,
} ..

Magick.NET Provider

`Volo.Abp.Imaging.MagickNet` NuGet package implements the image operations
using the [Magick.NET](https://github.com/dlemstra/Magick.NET) library.
```

#### ## Installation

You can add this package to your application by either using the [ABP CLI](CLI.md) or manually installing it. Using the [ABP CLI](CLI.md) is the recommended approach.

#### ### Using the ABP CLI

Open a command line terminal in the folder of your project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Imaging.MagickNet
````
```

#### ### Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.Imaging.MagickNet](https://www.nuget.org/packages/Volo.Abp.Imaging.MagickNet) NuGet package to your project:

```
```
Install-Package Volo.Abp.Imaging.MagickNet
````
```

2. Add `AbpImagingMagickNetModule` to your [module](Module-Development-Basics.md)'s dependency list:

```
```csharp
[DependsOn(typeof(AbpImagingMagickNetModule))]
public class MyModule : AbpModule
{
    //...
}
````
```

#### ### Configuration

`MagickNetCompressOptions` is an [options object](Options.md) that is used to configure the Magick.NET image compression system. It has the following properties:

```
* `OptimalCompression`: Indicates whether the optimal compression is enabled or not. (Default: `false`)
* `IgnoreUnsupportedFormats`: Indicates whether the unsupported formats are ignored or not. (Default: `false`)
* `Lossless`: Indicates whether the lossless compression is enabled or not. (Default: `false`)
```

#### ## ImageSharp Provider

`Volo.Abp.Imaging.ImageSharp` NuGet package implements the image operations using the [ImageSharp](<https://github.com/SixLabors/ImageSharp>) library.

#### ## Installation

You can add this package to your application by either using the [ABP CLI](CLI.md) or manually installing it. Using the [ABP CLI](CLI.md) is the recommended approach.

#### ### Using the ABP CLI

Open a command line terminal in the folder of your project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Imaging.ImageSharp
````
```

#### ### Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.Imaging.ImageSharp](<https://www.nuget.org/packages/Volo.Abp.Imaging.ImageSharp>) NuGet package to your project:

```
```
Install-Package Volo.Abp.Imaging.ImageSharp
````
```

2. Add `AbpImagingImageSharpModule` to your [module](Module-Development-Basics.md)'s dependency list:

```
```csharp
[DependsOn(typeof(AbpImagingImageSharpModule))]
public class MyModule : AbpModule
{
    //...
}
````
```

#### ### Configuration

`ImageSharpCompressOptions` is an [options object](Options.md) that is used to configure the ImageSharp image compression system. It has the following properties:

```
* `DefaultQuality`: The default quality of the JPEG and WebP encoders.
 (Default: `75`)
*
[`JpegEncoder`](https://docs.sixlabors.com/api/ImageSharp/SixLabors.ImageSharp.Formats.Jpeg.JpegEncoder.html): The JPEG encoder. (Default: `JpegEncoder` with `Quality` set to `DefaultQuality`)
*
[`PngEncoder`](https://docs.sixlabors.com/api/ImageSharp/SixLabors.ImageSharp.Formats.Png.PngEncoder.html): The PNG encoder. (Default: `PngEncoder` with `IgnoreMetadata` set to `true` and `CompressionLevel` set to `PngCompressionLevel.BestCompression`)
*
[`WebPEncoder`](https://docs.sixlabors.com/api/ImageSharp/SixLabors.ImageSharp.Formats.Webp.WebpEncoder.html): The WebP encoder. (Default: `WebPEncoder` with `Quality` set to `DefaultQuality`)
```

\*\*Example usage:\*\*

```
```csharp  
Configure<ImageSharpCompressOptions>(options =>  
{  
    options.JpegEncoder = new JpegEncoder  
    {  
        Quality = 60  
    };  
    options.PngEncoder = new PngEncoder  
    {  
        CompressionLevel = PngCompressionLevel.BestCompression  
    };  
    options.WebPEncoder = new WebPEncoder  
    {  
        Quality = 65  
    };  
});  
```
```

## ## ASP.NET Core Integration

`Volo.Abp.Imaging.AspNetCore` NuGet package defines attributes for controller actions that can automatically compress and/or resize uploaded files.

## ## Installation

You can add this package to your application by either using the [ABP CLI](CLI.md) or manually installing it. Using the [ABP CLI](CLI.md) is the recommended approach.

### ### Using the ABP CLI

Open a command line terminal in the folder of your project (.csproj file) and type the following command:

```
```bash  
abp add-package Volo.Abp.Imaging.AspNetCore  
```
```

### ### Manual Installation

If you want to manually install;

1. Add the [\[Volo.Abp.Imaging.AspNetCore\]\(https://www.nuget.org/packages/Volo.Abp.Imaging.AspNetCore\)](https://www.nuget.org/packages/Volo.Abp.Imaging.AspNetCore) NuGet package to your project:

...

`Install-Package Volo.Abp.Imaging.AspNetCore`

...

2. Add '`AbpImagingAspNetCoreModule`' to your `[module](Module-Development-Basics.md)`'s dependency list:

```
```csharp
[DependsOn(typeof(AbpImagingAspNetCoreModule))]
public class MyModule : AbpModule
{
    //...
}
```

```

### ### CompressImageAttribute

The '`CompressImageAttribute`' is used to compress the image before. '`IFormFile`', '`IRemoteStreamContent`', '`Stream`' and '`IEnumerable<byte>`' types are supported. It has the following properties:

- \* '`Parameters`': Names of the the parameters that are used to configure the image compression system. This is useful if your action has some non-image parameters. If you don't specify the parameters names, all of the method parameters are considered as image.

\*\*Example usage:\*\*

```
```csharp
[HttpPost]
[CompressImage] /* Compresses the given file (automatically determines the
file mime type) */
public async Task<IActionResult> Upload(IFormFile file)
{
    //...
}
```

```

### ### ResizeImageAttribute

The '`ResizeImageAttribute`' is used to resize the image before requesting the action. '`IFormFile`', '`IRemoteStreamContent`', '`Stream`' and '`IEnumerable<byte>`' types are supported. It has the following properties:

- \* '`Parameters`': Names of the the parameters that are used to configure the image resize system. This is useful if your action has some non-image parameters. If you don't specify the parameters names, all of the method parameters are considered as image.
- \* '`Width`': Target width of the resized image.
- \* '`Height`': Target height of the resized image.
- \* '`Mode`': The resize mode (see the `[ImageResizeMode](#imageresizemode)` section for more information).

**\*\*Example usage:\*\***

```
```csharp
[HttpPost]
[ResizeImage(Width = 100, Height = 100, Mode = ImageResizeMode.Crop)]
public async Task<IActionResult> Upload(IFormFile file)
{
    //...
}```
```

7.18 JSON

JSON

The ABP Framework provides an abstraction to work with JSON. Having such an abstraction has some benefits;

- * You can write library independent code. Therefore, you can change the underlying library with the minimum effort and code change.
- * You can use the predefined converters defined in the ABP without worrying about the underlying library's internal details.

> The JSON serialization system is implemented with the [\[Volo.Abp.Json\]](https://www.nuget.org/packages/Volo.Abp.Json)(<https://www.nuget.org/packages/Volo.Abp.Json>) NuGet package([[Volo.Abp.SystemTextJson](https://www.nuget.org/packages/Volo.Abp.SystemTextJson)](<https://www.nuget.org/packages/Volo.Abp.SystemTextJson>) is the default implementation). Most of the time, you don't need to manually [install it](#)(<https://abp.io/package-detail/Volo.Abp.Json>) since it comes pre-installed with the [\[application startup template\]](#)(Startup-Templates/Application.md).

IJsonSerializer

You can inject `IJsonSerializer` and use it for JSON operations. Here is the available operations in the `IJsonSerializer` interface.

```
```csharp
public interface IJsonSerializer
{
 string Serialize(object obj, bool camelCase = true, bool indented = false);
 T Deserialize<T>(string jsonString, bool camelCase = true);
 object Deserialize(Type type, string jsonString, bool camelCase = true);
}```
```

Usage Example:

```
```csharp
public class ProductManager
{
    public IJsonSerializer JsonSerializer { get; }

    public ProductManager(IJsonSerializer jsonSerializer)
    {
        JsonSerializer = jsonSerializer;
    }
}```
```

```

public void SendRequest(Product product)
{
    var json= JsonSerializer.Serialize(product);
    // Left blank intentionally for demo purposes...
}
```
Configuration

AbpJsonOptions

`'AbpJsonOptions'` type provides options for the JSON operations in the ABP Framework.

Properties:
* **InputDateFormats(`List<string>`)**: Formats of input JSON date, Empty string means default format. You can provide multiple formats to parse the date.
* **OutputDateFormat(`string`)**: Format of output json date, Null or empty string means default format.

System Text Json

AbpSystemTextJsonSerializerOptions

- **JsonSerializerOptions(`System.Text.Json.JsonSerializerOptions`)**: Global options for System.Text.Json library operations. See [here](https://docs.microsoft.com/en-us/dotnet/api/system.text.json.jsonserializeroptions) for reference.

AbpSystemTextJsonSerializerModifiersOptions

- **Modifiers(`List<Action<JsonTypeInfo>>`)**: Configure `Modifiers` of `DefaultJsonTypeInfoResolver`. See [here](https://devblogs.microsoft.com/dotnet/announcing-dotnet-7-preview-6/#json-contract-customization) for reference.

Newtonsoft

Add [Volo.Abp.Json.Newtonsoft](https://www.nuget.org/packages/Volo.Abp.Json.Newtonsoft) package and depends on `AbpJsonNewtonsoftModule` to replace the `System Text Json`.

AbpNewtonsoftJsonSerializerOptions

- **JsonSerializerSettings(`Newtonsoft.Json.JsonSerializerSettings`)**: Global options for Newtonsoft library operations. See [here](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonSerializerSettings.htm) for reference.

Configuring JSON options in ASP.NET Core

```

You can change the JSON behavior in ASP.NET Core by configuring [\[JsonOptions\]](https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.jsonoptions)(<https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.jsonoptions>) or [\[MvcNewtonsoftJsonOptions\]](https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.mvcnewtonsoftjsonoptions)(<https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.mvcnewtonsoftjsonoptions>)(if you use 'Newtonsoft.Json')

## 7.19 Object to Object Mapping

### # Object To Object Mapping

It's common to map an object to another similar object. It's also tedious and repetitive since generally both classes have the same or similar properties mapped to each other. Imagine a typical [\[application service\]](#)(Application-Services.md) method below:

```
```csharp
public class UserAppService : ApplicationService
{
    private readonly IRepository<User, Guid> _userRepository;

    public UserAppService(IRepository<User, Guid> userRepository)
    {
        _userRepository = userRepository;
    }

    public async Task CreateUser(CreateUserInput input)
    {
        //Manually creating a User object from the CreateUserInput object
        var user = new User
        {
            Name = input.Name,
            Surname = input.Surname,
            EmailAddress = input.EmailAddress,
            Password = input.Password
        };

        await _userRepository.InsertAsync(user);
    }
}..
```

'CreateUserInput' is a simple [\[DTO\]](#)(Data-Transfer-Objects.md) class and the 'User' is a simple [\[entity\]](#)(Entities.md). The code above creates a 'User' entity from the given input object. The 'User' entity will have more properties in a real-world application and manually creating it will become tedious and error-prone. You also have to change the mapping code when you add new properties to 'User' and 'CreateUserInput' classes.

We can use a library to automatically handle these kind of mappings. ABP provides abstractions for object to object mapping and has an integration package to use [\[AutoMapper\]](#)(<http://automapper.org/>) as the object mapper.

```
## IObjectMapper
```

`IObjectMapper` interface (in the [\[Volo.Abp.ObjectMapping\]](https://www.nuget.org/packages/Volo.Abp.ObjectMapping)(<https://www.nuget.org/packages/Volo.Abp.ObjectMapping>) package) defines a simple `Map` method. The example code introduced before can be re-written as shown below:

```
```csharp
public class UserAppService : ApplicationService
{
 private readonly IRepository<User, Guid> _userRepository;

 public UserAppService(IRepository<User, Guid> userRepository)
 {
 _userRepository = userRepository;
 }

 public async Task CreateUser(CreateUserInput input)
 {
 //Automatically creating a new User object using the CreateUserInput
 var user = ObjectMapper.Map<CreateUserInput, User>(input);

 await _userRepository.InsertAsync(user);
 }
}
````
```

> `ObjectMapper` is defined in the `ApplicationService` base class in this example. You can directly inject the `IObjectMapper` interface when you need it somewhere else.

Map method has two generic argument: First one is the source object type while the second one is the destination object type.

If you need to set properties of an existing object, you can use the second overload of the `Map` method:

```
```csharp
public class UserAppService : ApplicationService
{
 private readonly IRepository<User, Guid> _userRepository;

 public UserAppService(IRepository<User, Guid> userRepository)
 {
 _userRepository = userRepository;
 }

 public async Task UpdateUserAsync(Guid id, UpdateUserInput input)
 {
 var user = await _userRepository.GetAsync(id);

 //Automatically set properties of the user object using the
 UpdateUserInput
 ObjectMapper.Map<UpdateUserInput, User>(input, user);

 await _userRepository.UpdateAsync(user);
 }
}
````
```

You should have defined the mappings before to be able to map objects. See the AutoMapper integration section to learn how to define mappings.

AutoMapper Integration

[AutoMapper](http://automapper.org/) is one of the most popular object to object mapping libraries.

[Volo.Abp.AutoMapper](https://www.nuget.org/packages/Volo.Abp.AutoMapper) package defines the AutoMapper integration for the `IObjectMapper`.

Once you define mappings described as below, you can use the `IObjectMapper` interface just like explained before.

Define Mappings

AutoMapper provides multiple ways of defining mapping between classes. Refer to [its own documentation](https://docs.automapper.org) for all details.

One way to define object mappings is creating a [Profile](https://docs.automapper.org/en/stable/Configuration.html#profile-instances) class. Example:

```
```csharp
public class MyProfile : Profile
{
 public MyProfile()
 {
 CreateMap<User, UserDto>();
 }
}...```

```

You should then register profiles using the `AbpAutoMapperOptions`:

```
```csharp
[DependsOn(typeof(AbpAutoMapperModule))]
public class MyModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        Configure<AbpAutoMapperOptions>(options =>
        {
            //Add all mappings defined in the assembly of the MyModule class
            options.AddMaps<MyModule>();
        });
    }
}...```

```

`AddMaps` registers all profile classes defined in the assembly of the given class, typically your module class. It also registers for the [attribute mapping](https://docs.automapper.org/en/stable/Attribute-mapping.html).

Configuration Validation

`AddMaps` optionally takes a `bool` parameter to control the [configuration validation](https://docs.automapper.org/en/stable/Configuration-validation.html) for your [module](Module-Development-Basics.md):

```
```csharp
options.AddMaps<MyModule>(validate: true);
````
```

While this option is `false` by default, it is suggested to enable configuration validation as a best practice.

Configuration validation can be controlled per profile class using `AddProfile` instead of `AddMaps`:

```
```csharp
options.AddProfile<MyProfile>(validate: true);
````
```

> If you have multiple profiles and need to enable validation only for a few of them, first use `AddMaps` without validation, then use `AddProfile` for each profile you want to validate.

Mapping the Object Extensions

[Object extension system](Object-Extensions.md) allows to define extra properties for existing classes. ABP Framework provides a mapping definition extension to properly map extra properties of two objects.

```
```csharp
public class MyProfile : Profile
{
 public MyProfile()
 {
 CreateMap<User, UserDto>()
 .MapExtraProperties();
 }
}
````
```

It is suggested to use the `MapExtraProperties()` method if both classes are extensible objects (implement the `IHasExtraProperties` interface). See the [object extension document](Object-Extensions.md) for more.

Other Useful Extension Methods

There are some more extension methods those can simplify your mapping code.

Ignoring Audit Properties

It is common to ignore audit properties when you map an object to another.

Assume that you need to map a `ProductDto` ([DTO](Data-Transfer-Objects.md)) to a `Product` [entity](Entities.md) and the entity is inheriting from the `AuditedEntity` class (which provides properties like `CreationTime`, `CreatorId`, `IHasModificationTime`... etc).

You probably want to ignore these base properties while mapping from the DTO. You can use `'IgnoreAuditedObjectProperties()'` method to ignore all audit properties (instead of manually ignoring them one by one):

```
```csharp
public class MyProfile : Profile
{
 public MyProfile()
 {
 CreateMap<ProductDto, Product>()
 .IgnoreAuditedObjectProperties();
 }
}...```

```

There are more extension methods like `'IgnoreFullAuditedObjectProperties()'` and `'IgnoreCreationAuditedObjectProperties()'` those can be used based on your entity type.

> See the "*\*Base Classes & Interfaces for Audit Properties\**" section in the [\[entities document\]](#)(Entities.md) to know more about auditing properties.

#### #### Ignoring Other Properties

In AutoMapper, you typically write such a mapping code to ignore a property:

```
```csharp
public class MyProfile : Profile
{
    public MyProfile()
    {
        CreateMap<SimpleClass1, SimpleClass2>()
            .ForMember(x => x.CreationTime, map => map.Ignore());
    }
}...```

```

We found it unnecessarily long and created the `'Ignore()'` extension method:

```
```csharp
public class MyProfile : Profile
{
 public MyProfile()
 {
 CreateMap<SimpleClass1, SimpleClass2>()
 .Ignore(x => x.CreationTime);
 }
}...```

```

#### ## Advanced Topics

##### ### `IObjectMapper<TContext>` Interface

Assume that you have created a **\*\*reusable module\*\*** which defines AutoMapper profiles and uses `'IObjectMapper'` when it needs to map objects. Your module then can be used in different applications, by nature of the [\[modularity\]](#)(Module-Development-Basics.md).

`**IObjectMapper**` is an abstraction and can be replaced by the final application to use another mapping library. The problem here that your reusable module is designed to use the AutoMapper library, because it only defines mappings for it. In such a case, you will want to guarantee that your module always uses AutoMapper even if the final application uses another default object mapping library.

`**IObjectMapper<TContext>**` is used to contextualize the object mapper, so you can use different libraries for different modules/contexts.

Example usage:

```
```csharp
public class UserAppService : ApplicationService
{
    private readonly IRepository<User, Guid> _userRepository;
    private readonly IObjectMapper<MyModule> _objectMapper;

    public UserAppService(
        IRepository<User, Guid> userRepository,
        IObjectMapper<MyModule> objectMapper) //Inject module specific mapper
    {
        _userRepository = userRepository;
        _objectMapper = objectMapper;
    }

    public async Task CreateUserAsync(CreateUserInput input)
    {
        //Use the module specific mapper
        var user = _objectMapper.Map<CreateUserInput, User>(input);

        await _userRepository.InsertAsync(user);
    }
}
```

```

`UserAppService` injects the `IObjectMapper<MyModule>`, the specific object mapper for this module. It's usage is exactly same of the `IObjectMapper` .

The example code above don't use the `ObjectMapper` property defined in the `ApplicationService` , but injects the `IObjectMapper<MyModule>` . However, it is still possible to use the base property since the `ApplicationService` defines an `ObjectContext` property that can be set in the class constructor. So, the example about can be re-written as like below:

```
```csharp
public class UserAppService : ApplicationService
{
    private readonly IRepository<User, Guid> _userRepository;

    public UserAppService(IRepository<User, Guid> userRepository)
    {
        _userRepository = userRepository;
        //Set the object mapper context
        ObjectMapperContext = typeof(MyModule);
    }
}
```

```

```

public async Task CreateUserAsync(CreateUserInput input)
{
 var user = ObjectMapper.Map<CreateUserInput, User>(input);

 await _userRepository.InsertAsync(user);
}
...

```

While using the contextualized object mapper is same as the normal object mapper, you should register the contextualized mapper in your module's `'ConfigureServices'` method:

```

```csharp
[DependsOn(typeof(AbpAutoMapperModule))]
public class MyModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        //Use AutoMapper for MyModule
        context.Services.AddAutoMapperObjectMapper<MyModule>();

        Configure<AbpAutoMapperOptions>(options =>
        {
            options.AddMaps<MyModule>(validate: true);
        });
    }
}
...

```

`'IObjectMapper<MyModule>'` is an essential feature for a reusable module where it can be used in multiple applications each may use a different library for object to object mapping. All pre-built ABP modules are using it. But, for the final application, you can ignore this interface and always use the default `'IObjectMapper'` interface.

`IObjectMapper<TSource, TDestination>` Interface

ABP allows you to customize the mapping code for specific classes. Assume that you want to create a custom class to map from `'User'` to `'UserDto'`. In this case, you can create a class that implements the `'IObjectMapper<User, UserDto>'`:

```

```csharp
public class MyCustomUserMapper : IObjectMapper<User, UserDto>,
ITransientDependency
{
 public UserDto Map(User source)
 {
 //TODO: Create a new UserDto
 }

 public UserDto Map(User source, UserDto destination)
 {
 //TODO: Set properties of an existing UserDto
 return destination;
 }
}

```

```
 }
}
```

ABP automatically discovers and registers the `MyCustomUserMapper` and it is automatically used whenever you use the `IObjectMapper` to map `User` to `UserDto`. A single class may implement more than one `IObjectMapper<TSource, TDestination>` each for a different object pairs.

> This approach is powerful since `MyCustomUserMapper` can inject any other service and use in the `Map` methods.

Once you implement `IObjectMapper<User, UserDto>`, ABP can automatically convert a collection of `User` objects to a collection of `UserDto` objects. The following generic collection types are supported:

```
* `IEnumerable<T>`
* `ICollection<T>`
* `Collection<T>`
* ` IList<T>`
* `List<T>`
* `T[]` (array)
```

**\*\*Example:\*\***

```
```csharp
var users = await _userRepository.GetListAsync(); // returns List<User>
var dtos = ObjectMapper.Map<List<User>, List<UserDto>>(users); // creates
List<UserDto>
```
```

## 7.20 Simple State Checker

### # Simple State Checker

The simple state checking system can be used to enable/disable an object based on some dynamic conditions. For example, you can disable a menu item on the user interface, if the current user has not granted for a given permission. The simple state checking system provides a generic way to define and check such conditions.

#### ## Definition state checker.

Any class can inherit `IHasSimpleStateCheckers` to support state checks.

```
```csharp
public class MyObject : IHasSimpleStateCheckers<MyObject>
{
    public int Id { get; set; }

    public List<ISimpleStateChecker<MyObject>> SimpleStateCheckers { get; }

    public MyObject()
    {
        SimpleStateCheckers = new List<ISimpleStateChecker<MyObject>>();
    }
}
```

```
} ...
```

The `MyObject` class contains a collection of state checkers, you can add your custom checkers to it.

```
```csharp
public class MyObjectStateChecker : ISimpleStateChecker<MyObject>
{
 public Task<bool> IsEnabledAsync(SimpleStateCheckerContext<MyObject> context)
 {
 var currentUser =
context.ServiceProvider.GetRequiredService<ICurrentUser>();
 return Task.FromResult(currentUser.IsInRole("Admin"));
 }
} ...
```
```csharp
var myobj = new MyObject()
{
 Id = 100
};

myobj.SimpleStateCheckers.Add(new MyObjectStateChecker());
```

```

Definition Global State Checkers

`AbpSimpleStateCheckerOptions` is the options class that used to set the global state checkers for specific object.

Example: Add global state for `MyObject`:

```
```csharp
services.Configure<AbpSimpleStateCheckerOptions<MyObject>>(options =>
{
 options.GlobalSimpleStateCheckers.Add<MyGlobalObjectStateChecker>();
 //options.GlobalSimpleStateCheckers.Add<>(); //Add more global state checkers
});
```

```

> Write this inside the `ConfigureServices` method of your module.

Check the state

You can inject `ISimpleStateCheckerManager<MyObject>` service to check state.

```
```csharp
bool enabled = await stateCheckerManager.IsEnabledAsync(myobj);
```

```

Batch check the states

If there are many instance items that require state checking, there may be performance problems.

In this case, you can implement `ISimpleBatchStateChecker`. It can check multiple items at once.

You need to make sure that the same `ISimpleBatchStateChecker` instance is added to the `SimpleStateCheckers` of multiple instances.

> `SimpleBatchStateCheckerBase` inherits the `ISimpleBatchStateChecker` interface and implements the `IsEnabledAsync` method of a single object by default.

```
```csharp
public class MyObjectBatchStateChecker :
SimpleBatchStateCheckerBase<MyObject>
{
 public override Task<SimpleStateCheckerResult<MyObject>>
IsEnabledAsync(SimpleBatchStateCheckerContext<MyObject> context)
{
 var result = new SimpleStateCheckerResult<MyObject>(context.States);

 foreach (var myObject in context.States)
 {
 if (myObject.Id > 100)
 {
 result[myObject] = true;
 }
 }

 return Task.FromResult(result);
}
}...
```
```csharp
var myobj1 = new MyObject()
{
 Id = 100
};
var myobj2 = new MyObject()
{
 Id = 99
};

var myObjectBatchStateChecker = new MyObjectBatchStateChecker();

myobj1.SimpleStateCheckers.Add(myObjectBatchStateChecker);
myobj2.SimpleStateCheckers.Add(myObjectBatchStateChecker);

SimpleStateCheckerResult<MyObject> stateCheckerResult = await
stateCheckerManager.IsEnabledAsync(new []{ myobj1, myobj2 });
```

```

Built-in State Checkers

The `PermissionDefinition`, `ApplicationMenuItem` and `ToolbarItem` objects have implemented state checks and have built-in general state checkers, you can directly use their extension methods.

```
```csharp
```

```
RequireAuthenticated();
RequirePermissions(bool requiresAll, params string[] permissions);
RequireFeatures(bool requiresAll, params string[] features);
RequireGlobalFeatures(bool requiresAll, params Type[] globalFeatures);
```
```

7.21 SMS Sending

SMS Sending

The ABP Framework provides an abstraction to sending SMS. Having such an abstraction has some benefits;

- You can then **easily change** your SMS sender without changing your application code.
- If you want to create **reusable application modules**, you don't need to make assumption about how the SMS are sent.

Installation

It is suggested to use the [ABP CLI](CLI.md) to install this package.

Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Sms
````
```

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.Sms).

Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.Sms](https://www.nuget.org/packages/Volo.Abp.Sms) NuGet package to your project:

```
```
```

```
Install-Package Volo.Abp.Sms
````
```

2. Add the `AbpSmsModule` to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpSmsModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
```

...

## ## Sending SMS

[Inject](Dependency-Injection.md) the `ISmsSender` into any service and use the `SendAsync` method to send a SMS.

\*\*Example:\*\*

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Sms;

namespace MyProject
{
    public class MyService : ITransientDependency
    {
        private readonly ISmsSender _smsSender;

        public MyService(ISmsSender smsSender)
        {
            _smsSender = smsSender;
        }

        public async Task DoItAsync()
        {
            await _smsSender.SendAsync(
                "+012345678901",           // target phone number
                "This is test sms..."     // message text
            );
        }
    }
}..
```

The given `SendAsync` method in the example is an extension method to send an SMS with primitive parameters. In addition, you can pass an `SmsMessage` object which has the following properties:

- `PhoneNumber` (`string`): Target phone number
- `Text` (`string`): Message text
- `Properties` (`Dictionary<string, string>`): Key-value pairs to pass custom arguments

NullSmsSender

`NullSmsSender` is a the default implementation of the `ISmsSender`. It writes SMS content to the [standard logger](Logging.md), rather than actually sending the SMS.

This class can be useful especially in development time where you generally don't want to send real SMS. **However, if you want to actually send SMS, you should implement the `ISmsSender` in your application code.**

Implementing the ISmsSender

You can easily create your SMS sending implementation by creating a class that implements the `ISmsSender` interface, as shown below:

```
```csharp
using System.IO;
using System.Threading.Tasks;
using Volo.Abp.Sms;
using Volo.Abp.DependencyInjection;

namespace AbpDemo
{
 public class MyCustomSmsSender : ISmsSender, ITransientDependency
 {
 public async Task SendAsync(SmsMessage smsMessage)
 {
 // Send sms
 }
 }
}

More

[ABP Commercial](https://commercial.abp.io/) provides Twilio integration package to send SMS over [Twilio service](https://docs.abp.io/en/commercial/latest/modules/twilio-sms).
```

## 7.22 String Encryption

### # String Encryption

ABP Framework provides string encryption feature that allows to **Encrypt** and **Decrypt** strings.

#### ## Installation

> This package is already installed by default with the startup template. So, most of the time, you don't need to install it manually.

If installation is needed, it is suggested to use the [\[ABP CLI\]](#)(https://docs.abp.io/en/abp/latest/CLI) to install this package.

#### ### Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Security
```

```

#### ### Manual Installation

If you want to manually install;

1. Add the [\[Volo.Abp.Security\]\(https://www.nuget.org/packages/Volo.Abp.Security\)](https://www.nuget.org/packages/Volo.Abp.Security) NuGet package to your project:

```
'Install-Package Volo.Abp.Security'
```

2. Add the `AbpSecurityModule` to the dependency list of your module:

```
```csharp
[DependsOn(
    //...other dependencies
    typeof(AbpSecurityModule) // <-- Add module dependency like that
)]
public class YourModule : AbpModule
{
}
```

```

#### ## Using String Encryption

All encryption operations are included in `IStringEncryptionService`. You can inject it and start to use.

```
```csharp
public class MyService : DomainService
{
    protected IStringEncryptionService StringEncryptionService { get; }

    public MyService(IStringEncryptionService stringEncryptionService)
    {
        StringEncryptionService = stringEncryptionService;
    }

    public string Encrypt(string value)
    {
        // To encrypt a value
        return StringEncryptionService.Encrypt(value);
    }

    public string Decrypt(string value)
    {
        // To decrypt a value
        return StringEncryptionService.Decrypt(value);
    }
}
```

```

#### ### Using Custom PassPhrase

`IStringEncryptionService` methods has \*\*passPhrase\*\* parameter with default value and it uses default PassPhrase when you don't pass passPhrase parameter.

```
```csharp
// Default Pass Phrase
var encryptedValue = StringEncryptionService.Encrypt(value);

// Custom Pass Phrase

```

```

var encryptedValue = StringEncryptionService.Encrypt(value,
"MyCustomPassPhrase");

// Encrypt & Decrypt have same parameters.
var decryptedValue = StringEncryptionService.Decrypt(value,
"MyCustomPassPhrase");
```

Using Custom Salt

`IStringEncryptionService` methods has **salt** parameter with default value and it uses default Salt when you don't pass the parameter.

```csharp
// Default Salt
var encryptedValue = StringEncryptionService.Encrypt(value);

// Custom Salt
var encryptedValue = StringEncryptionService.Encrypt(value, salt:
Encoding.UTF8.GetBytes("MyCustomSalt"));

// Encrypt & Decrypt have same parameters.
var decryptedValue = StringEncryptionService.Decrypt(value, salt:
Encoding.UTF8.GetBytes("MyCustomSalt"));
```

String Encryption Options

Default values can be configured with `AbpStringEncryptionOptions` type.

```csharp
Configure<AbpStringEncryptionOptions>(opts =>
{
    opts.DefaultPassPhrase = "MyStrongPassPhrase";
    opts.DefaultSalt = Encoding.UTF8.GetBytes("MyStrongSalt");
    opts.InitVectorBytes = Encoding.UTF8.GetBytes("YetAnotherStrongSalt");
    opts.Keysize = 512;
});
```

- **DefaultPassPhrase**: Default password to encrypt/decrypt texts. It's recommended to set to another value for security. Default value: `gsKnGZ041HLL4IM8`

- **DefaultSalt**: A value which is used as salt while encrypting/decrypting.

 Default value: `Encoding.ASCII.GetBytes("hgt!16kl")`

- **InitVectorBytes**: This constant string is used as a "salt" value for the PasswordDeriveBytes function calls. This size of the IV (in bytes) must = (keysize / 8). Default keysize is 256, so the IV must be 32 bytes long. Using a 16 character string here gives us 32 bytes when converted to a byte array.

 Default value: `Encoding.ASCII.GetBytes("jkE49230Tf093b42")`

```

- **\*\*Keysize:\*\*** This constant is used to determine the keysize of the encryption algorithm.

Default value: `256`

## 7.23 Text Templating

# Text Templating

## Introduction

ABP Framework provides a simple, yet efficient text template system. Text templating is used to dynamically render contents based on a template and a model (a data object):

Template + Model =renderer=> Rendered Content

It is very similar to an ASP.NET Core Razor View (or Page):

\*RAZOR VIEW (or PAGE) + MODEL ==render==> HTML CONTENT\*

You can use the rendered output for any purpose, like sending emails or preparing some reports.

Template rendering engine is very powerful;

- \* It supports **conditional logics**, **loops** and much more.
- \* Template content **can be localized**.
- \* You can define **layout templates** to be used as the layout while rendering other templates.
- \* You can pass arbitrary objects to the template context (beside the model) for advanced scenarios.

ABP Framework provides two templating engines;

- \* **[Razor](Text-Templating-Razor.md)**
- \* **[Scriban](Text-Templating-Scriban.md)**

You can use different template engines in the same application, or even create a new custom template engine.

## Source Code

Get [the source code of the sample application](<https://github.com/abpframework/abp-samples/tree/master/TextTemplateDemo>) developed and referred through this document.

## See Also

- \* [The source code of the sample application](<https://github.com/abpframework/abp-samples/tree/master/TextTemplateDemo>) developed and referred through this document.
- \* [Localization system](Localization.md).
- \* [Virtual File System](Virtual-File-System.md).

### 7.23.1 Razor Integration

#### # Razor Integration

The Razor template is a standard C# class, so you can freely use the functions of C#, such as '[dependency injection](#)', using '[LINQ](#)', custom methods, and even using '[Repository](#)'.

#### ## Installation

It is suggested to use the [\[ABP CLI\]\(CLI.md\)](#) to install this package.

#### ### Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.TextTemplating.Razor
````
```

> If you haven't done it yet, you first need to install the [\[ABP CLI\]\(CLI.md\)](#). For other installation options, see [\[the package description page\]\(https://abp.io/package-detail/Volo.Abp.TextTemplating.Razor\)](#).

#### ### Manual Installation

If you want to manually install;

1. Add the [\[Volo.Abp.TextTemplating.Razor\]\(https://www.nuget.org/packages/Volo.Abp.TextTemplating.Razor\)](#) NuGet package to your project:

```
```
Install-Package Volo.Abp.TextTemplating.Razor
````
```

2. Add the '[`AbpTextTemplatingRazorModule`](#)' to the dependency list of your module:

```
```csharp
[DependsOn(
    //...other dependencies
    typeof(AbpTextTemplatingRazorModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
````
```

#### ## Add MetadataReference to CSharpCompilerOptions

You need to add the '[`MetadataReference`](#)' of the type used in the template to '[`CSharpCompilerOptions's References`](#)'.

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    Configure<AbpRazorTemplateCSharpCompilerOptions>(options =>
    {
        options.References.Add(MetadataReference.CreateFromFile(typeof(YourModule).Assembly.Location));
    });
}
```

```

## Add MetadataReference for a template.

You can add some `MetadataReference` to the template

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    services.Configure<AbpCompiledViewProviderOptions>(options =>
    {
        //Hello is template name.
        options.TemplateReferences.Add("Hello", new List<Assembly>()
        {
            Assembly.Load("Microsoft.Extensions.Logging.Abstractions"),
            Assembly.Load("Microsoft.Extensions.Logging")
        }
        .Select(x => MetadataReference.CreateFromFile(x.Location))
        .ToList());
    });
}
```

```

## Defining Templates

Before rendering a template, you should define it. Create a class inheriting from the `TemplateDefinitionProvider` base class:

```
```csharp
public class DemoTemplateDefinitionProvider : TemplateDefinitionProvider
{
    public override void Define(ITemplateDefinitionContext context)
    {
        context.Add(
            new TemplateDefinition("Hello") //template name: "Hello"
            .WithRazorEngine()
            .WithVirtualFilePath(
                "/Demos/Hello/Hello.cshtml", //template content path
                isInlineLocalized: true
            )
        );
    }
}
```

```

\* `context` object is used to add new templates or get the templates defined by depended modules. Used `context.Add(...)` to define a new template.

```
* `TemplateDefinition` is the class represents a template. Each template must have a unique name (that will be used while you are rendering the template).
* `/Demos/Hello/Hello.cshtml` is the path of the template file.
* `isInlineLocalized` is used to declare if you are using a single template for all languages (`true`) or different templates for each language (`false`). See the Localization section below for more.
* `WithRenderEngine` method is used to set the render engine of the template.
```

### ### The Template Base

Every `cshtml` template page needs to inherit `RazorTemplatePageBase` or `RazorTemplatePageBase<Model>`.

There are some useful properties in the base class that can be used in templates. eg: `Localizer`, `ServiceProvider`.

### ### The Template Content

`WithVirtualFilePath` indicates that we are using the [Virtual File System](Virtual-File-System.md) to store the template content. Create a `Hello.cshtml` file inside your project and mark it as "embedded resource" on the properties window:

```
![hello-template-razor](images/hello-template-razor.png)
```

Example `Hello.cshtml` content is shown below:

```
....
@inherits
Volo.Abp.TextTemplating.Razor.RazorTemplatePageBase<HelloModelNamespace.Hello
Model>
Hello @Model.Name
....
```

The `HelloModel` class is:

```
```csharp  
namespace HelloModelNamespace  
{  
    public class HelloModel  
    {  
        public string Name { get; set; }  
    }  
}...  
....
```

The [Virtual File System](Virtual-File-System.md) requires to add your files in the `ConfigureServices` method of your [module](Module-Development-Basics.md) class:

```
```csharp  
Configure<AbpVirtualFileSystemOptions>(options =>
{
 options.FileSets.AddEmbedded<TextTemplateDemoModule>("TextTemplateDemo");
});
....
```

\* `TextTemplateDemoModule` is the module class that you define your template in.  
\* `TextTemplateDemo` is the root namespace of your project.

```

Rendering the Template

`ITemplateRenderer` service is used to render a template content.

Example: Rendering a Simple Template

```csharp
public class HelloDemo : ITransientDependency
{
    private readonly ITemplateRenderer _templateRenderer;

    public HelloDemo(ITemplateRenderer templateRenderer)
    {
        _templateRenderer = templateRenderer;
    }

    public async Task RunAsync()
    {
        var result = await _templateRenderer.RenderAsync(
            "Hello", //the template name
            new HelloModel
            {
                Name = "John"
            }
        );
        Console.WriteLine(result);
    }
}
```
* `HelloDemo` is a simple class that injects the `ITemplateRenderer` in its constructor and uses it inside the `RunAsync` method.
* `RenderAsync` gets two fundamental parameters:
 * `templateName`: The name of the template to be rendered (`Hello` in this example).
 * `model`: An object that is used as the `model` inside the template (a `HelloModel` object in this example).

```

The result shown below for this example:

```

```csharp
Hello John :)
```

Localization

```

It is possible to localize a template content based on the current culture. There are two types of localization options described in the following sections.

#### ### Inline localization

Inline localization uses the [localization system](Localization.md) to localize texts inside templates.

#### #### Example: Reset Password Link

Assuming you need to send an email to a user to reset her/his password. Here, the model/template content:

```
```csharp
namespace ResetMyPasswordModelNamespace
{
    public class ResetMyPasswordModel
    {
        public string Link { get; set; }

        public string Name { get; set; }
    }
}...```
```html
@inherits
Volo.Abp.TextTemplating.Razor.RazorTemplatePageBase<ResetMyPasswordModelNames
pace.ResetMyPasswordModel>
<a title="@Localizer["ResetMyPasswordTitle"]"
href="@Model.Link">@Localizer["ResetMyPassword", Model.Name]
```

```

`Localizer` service is used to localize the given key based on the current user culture. You need to define the `ResetMyPassword` key inside your localization file:

```
```json
"ResetMyPasswordTitle": "Reset my password",
"ResetMyPassword": "Hi {0}, Click here to reset your password"
```

```

You also need to declare the localization resource to be used with this template, inside your template definition provider class:

```
```csharp
context.Add(
 new TemplateDefinition(
 "PasswordReset", //Template name
 typeof(DemoResource) //LOCALIZATION RESOURCE
)
 .WithRazorEngine()
 .WithVirtualFilePath(
 "/Demos/PasswordReset/PasswordReset.cshtml", //template content
path
 isInlineLocalized: true
)
);...```

```

That's all. When you render this template like that:

```
```csharp
var result = await _templateRenderer.RenderAsync(
    "PasswordReset", //the template name
    new PasswordResetModel
    {
        Name = "john",

```

```
        Link = "https://abp.io/example-link?userId=123&token=ABC"
    }
};
```

You will see the localized result:

```
```html
<a title="Reset my password" href="https://abp.io/example-
link?userId=123&token=ABC">Hi john, Click here to reset your password
````
```

> If you define the [default localization resource](Localization.md) for your application, then no need to declare the resource type for the template definition.

Multiple Contents Localization

Instead of a single template that uses the localization system to localize the template, you may want to create different template files for each language. It can be needed if the template should be completely different for a specific culture rather than simple text localizations.

Example: Welcome Email Template

Assuming that you want to send a welcome email to your users, but want to define a completely different template based on the user culture.

First, create a folder and put your templates inside it, like `en.cshtml`, `tr.cshtml`... one for each culture you support:

```
![multiple-file-template-razor](images/multiple-file-template-razor.png)
```

Then add your template definition in the template definition provider class:

```
```csharp
context.Add(
 new TemplateDefinition(
 name: "WelcomeEmail",
 defaultCultureName: "en"
)
 .WithRazorEngine()
 .WithVirtualFilePath(
 "/Demos/WelcomeEmail/Templates", //template content folder
 isInlineLocalized: false
)
);
```

\* Set \*\*default culture name\*\*, so it fallbacks to the default culture if there is no template for the desired culture.  
\* Specify \*\*the template folder\*\* rather than a single template file.  
\* Set `isInlineLocalized` to `false` for this case.

That's all, you can render the template for the current culture:

```
```csharp
var result = await _templateRenderer.RenderAsync("WelcomeEmail");
```

....

> Skipped the modal for this example to keep it simple, but you can use models as just explained before.

Specify the Culture

'ITemplateRenderer' service uses the current culture ('CultureInfo.CurrentCulture') if not specified. If you need, you can specify the culture as the 'cultureName' parameter:

```
```csharp
var result = await _templateRenderer.RenderAsync(
 "WelcomeEmail",
 cultureName: "en"
);```

```

### ## Layout Templates

Layout templates are used to create shared layouts among other templates. It is similar to the layout system in the ASP.NET Core MVC / Razor Pages.

### ### Example: Email HTML Layout Template

For example, you may want to create a single layout for all of your email templates.

First, create a template file just like before:

```
```html
@inherits Volo.Abp.TextTemplating.Razor.RazorTemplatePageBase
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
</head>
<body>
    @Body
</body>
</html>
```

```

\* A layout template must have a 'Body' part as a place holder for the rendered child content.

The register your template in the template definition provider:

```
```csharp
context.Add(
    new TemplateDefinition(
        "EmailLayout",
        isLayout: true //SET isLayout!
    )
    .WithRazorEngine()
    .WithVirtualFilePath(
        "/Demos/EmailLayout/EmailLayout.cshtml",
        isInlineLocalized: true
)
```

```

```
)
);
};
```

Now, you can use this template as the layout of any other template:

```
```csharp  
context.Add(  
    new TemplateDefinition(  
        name: "WelcomeEmail",  
        defaultCultureName: "en",  
        layout: "EmailLayout" //Set the LAYOUT  
    )  
    .WithRazorEngine()  
    .WithVirtualFilePath(  
        "/Demos/WelcomeEmail/Templates",  
        isInlineLocalized: false  
    )  
);  
};
```

Global Context

ABP passes the `model` that can be used to access to the model inside the template. You can pass more global variables if you need.

An example template content:

```
```html  
@inherits Volo.Abp.TextTemplating.Razor.RazorTemplatePageBase
A global object value: @GlobalContext["myGlobalObject"]
```
```

This template assumes that that is a `myGlobalObject` object in the template rendering context. You can provide it like shown below:

```
```csharp  
var result = await _templateRenderer.RenderAsync(
 "GlobalContextUsage",
 globalContext: new Dictionary<string, object>
 {
 {"myGlobalObject", "TEST VALUE"}
 }
);
};
```

The rendering result will be:

```
```  
A global object value: TEST VALUE  
```
```

#### ## Replacing the Existing Templates

It is possible to replace a template defined by a module that used in your application. In this way, you can customize the templates based on your requirements without changing the module code.

### ### Option-1: Using the Virtual File System

The [Virtual File System](Virtual-File-System.md) allows you to override any file by placing the same file into the same path in your project.

#### #### Example: Replace the Standard Email Layout Template

ABP Framework provides an [email sending system](Emailing.md) that internally uses the text templating to render the email content. It defines a standard email layout template in the `/Volo/Abp/Emailing/Templates/Layout.cshtml` path. The unique name of the template is `Abp.StandardEmailTemplates.Layout` and this string is defined as a constant on the `Volo.Abp.Emailing.Templates.StandardEmailTemplates` static class.

Do the following steps to replace the template file with your own;

\*\*1)\*\* Add a new file into the same location (`/Volo/Abp/Emailing/Templates/Layout.cshtml`) in your project:

```
![replace-email-layout-razor](images/replace-email-layout-razor.png)
```

\*\*2)\*\* Prepare your email layout template:

```
```html
@inherits Volo.Abp.TextTemplating.Razor.RazorTemplatePageBase
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
</head>
<body>
    <h1>This my header</h1>

    @Body

    <footer>
        This is my footer...
    </footer>
</body>
</html>
```
```

This example simply adds a header and footer to the template and renders the content between them (see the \*Layout Templates\* section above to understand it).

\*\*3)\*\* Configure the embedded resources in the `\*.csproj` file

- \* Add [Microsoft.Extensions.FileProviders.Embedded](https://www.nuget.org/packages/Microsoft.Extensions.FileProviders.Embedded) NuGet package to the project.
- \* Add `<GenerateEmbeddedFilesManifest>true</GenerateEmbeddedFilesManifest>` into the `<PropertyGroup>...</PropertyGroup>` section of your `\*.csproj` file.
- \* Add the following code into your `\*.csproj` file:

```
```xml
<ItemGroup>
    <None Remove="Volo\Abp\Emailing\Templates\*.cshtml" />

```

```
<EmbeddedResource Include="Volo\Abp\Emailing\Templates\*.cshtml" />
</ItemGroup>
````
```

This makes the template files "embedded resource".

\*\*4)\*\* Configure the virtual file system

Configure the `AbpVirtualFileSystemOptions` in the `ConfigureServices` method of your [module](Module-Development-Basics.md) to add the embedded files into the virtual file system:

```
```csharp
Configure<AbpVirtualFileSystemOptions>(options =>
{
    options.FileSets.AddEmbedded<BookStoreDomainModule>();
});
```

`BookStoreDomainModule` should be your module name, in this example code.

> Be sure that your module (directly or indirectly) [depends on](Module-Development-Basics.md) the `AbpEmailingModule`. Because the VFS can override files based on the dependency order.

Now, your template will be used when you want to render the email layout template.

Option-2: Using the Template Definition Provider

You can create a template definition provider class that gets the email layout template and changes the virtual file path for the template.

Example: Use the `/MyTemplates/EmailLayout.cshtml` file instead of the standard template

```
```csharp
using Volo.Abp.DependencyInjection;
using Volo.Abp.Emailing.Templates;
using Volo.Abp.TextTemplating;

namespace MyProject
{
 public class MyTemplateDefinitionProvider
 : TemplateDefinitionProvider, ITransientDependency
 {
 public override void Define(ITemplateDefinitionContext context)
 {
 var emailLayoutTemplate =
context.GetOrNull(StandardEmailTemplates.Layout);

 emailLayoutTemplate
 .WithVirtualFilePath(
 "/MyTemplates/EmailLayout.cshtml",
 isInlineLocalized: true
);
 }
 }
}
```

```
} ..
```

You should still add the file `'/MyTemplates/EmailLayout.cshtml` to the virtual file system as explained before. This approach allows you to locate templates in any folder instead of the folder defined by the depended module.

Beside the template content, you can manipulate the template definition properties, like `DisplayName`, `Layout` or `LocalizationSource`.

## ## Advanced Features

This section covers some internals and more advanced usages of the text templating system.

### ### Template Content Provider

`ITemplateRenderer` is used to render the template, which is what you want for most of the cases. However, you can use the `ITemplateContentProvider` to get the raw (not rendered) template contents.

> `ITemplateContentProvider` is internally used by the `ITemplateRenderer` to get the raw template contents.

Example:

```
```csharp
public class TemplateContentDemo : ITransientDependency
{
    private readonly ITemplateContentProvider _templateContentProvider;

    public TemplateContentDemo(ITemplateContentProvider
templateContentProvider)
    {
        _templateContentProvider = templateContentProvider;
    }

    public async Task RunAsync()
    {
        var result = await _templateContentProvider
            .GetContentOrNullAsync("Hello");

        Console.WriteLine(result);
    }
}...
```

The result will be the raw template content:

```
...
@inherits Volo.Abp.TextTemplating.Razor.RazorTemplatePageBase<HelloModelNamespace.Hello
Model>
Hello @Model.Name
...``
```

* `GetContentOrNullAsync` returns `null` if no content defined for the requested template.

* It can get a `cultureName` parameter that is used if template has different files for different cultures (see [Multiple Contents Localization](#) section above).

Template Content Contributor

`ITemplateContentProvider` service uses `ITemplateContentContributor` implementations to find template contents. There is a single pre-implemented content contributor, `VirtualFileTemplateContentContributor`, which gets template contents from the virtual file system as described above.

You can implement the `ITemplateContentContributor` to read raw template contents from another source.

Example:

```
```csharp
public class MyTemplateContentProvider
 : ITemplateContentContributor, ITransientDependency
{
 public async Task<string>
GetOrNullAsync(TemplateContentContributorContext context)
 {
 var templateName = context.TemplateDefinition.Name;

 //TODO: Try to find content from another source
 return null;
 }
}
```
....
```

Return `null` if your source can not find the content, so `ITemplateContentProvider` fallbacks to the next contributor.

Template Definition Manager

`ITemplateDefinitionManager` service can be used to get the template definitions (created by the template definition providers).

See Also

- * [\[The source code of the sample application\]](#)(<https://github.com/abpframework/abp-samples/tree/master/TextTemplateDemo>) developed and referred through this document.
- * [\[Localization system\]](#)(Localization.md).
- * [\[Virtual File System\]](#)(Virtual-File-System.md).

7.23.2 Scriban Integration

Scriban Integration

Installation

It is suggested to use the [\[ABP CLI\]](#)(CLI.md) to install this package.

Using the ABP CLI

Open a command line window in the folder of the project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.TextTemplating.Scriban
````
```

Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.TextTemplating.Scriban](<https://www.nuget.org/packages/Volo.Abp.TextTemplating.Scriban>) NuGet package to your project:

```
```
Install-Package Volo.Abp.TextTemplating.Scriban
````
```

2. Add the `AbpTextTemplatingScribanModule` to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpTextTemplatingScribanModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
````
```

Defining Templates

Before rendering a template, you should define it. Create a class inheriting from the `TemplateDefinitionProvider` base class:

```
```csharp
public class DemoTemplateDefinitionProvider : TemplateDefinitionProvider
{
 public override void Define(ITemplateDefinitionContext context)
 {
 context.Add(
 new TemplateDefinition("Hello") //template name: "Hello"
 .WithVirtualFilePath(
 "/Demos/Hello/Hello.tpl", //template content path
 isInlineLocalized: true
)
 .WithScribanEngine()
);
 }
}
````
```

```
* `context` object is used to add new templates or get the templates defined by depended modules. Used `context.Add(...)` to define a new template.
* `TemplateDefinition` is the class represents a template. Each template must have a unique name (that will be used while you are rendering the template).
* `/Demos/Hello/Hello.tpl` is the path of the template file.
* `isInlineLocalized` is used to declare if you are using a single template for all languages (`true`) or different templates for each language (`false`). See the Localization section below for more.
* `WithRenderEngine` method is used to set the render engine of the template.
```

The Template Content

`WithVirtualFilePath` indicates that we are using the [Virtual File System](Virtual-File-System.md) to store the template content. Create a `Hello.tpl` file inside your project and mark it as "embedded resource" on the properties window:

```
![hello-template](images/hello-template.png)
```

Example `Hello.tpl` content is shown below:

```
```
Hello {{model.name}} :)
```

The [Virtual File System](Virtual-File-System.md) requires to add your files in the `ConfigureServices` method of your [module](Module-Development-Basics.md) class:

```
```csharp
Configure<AbpVirtualFileSystemOptions>(options =>
{
    options.FileSets.AddEmbedded<TextTemplateDemoModule>("TextTemplateDemo");
});
```

* `TextTemplateDemoModule` is the module class that you define your template in.
* `TextTemplateDemo` is the root namespace of your project.

Rendering the Template

`ITemplateRenderer` service is used to render a template content.

Example: Rendering a Simple Template

```
```csharp
public class HelloDemo : ITransientDependency
{
 private readonly ITemplateRenderer _templateRenderer;

 public HelloDemo(ITemplateRenderer templateRenderer)
 {
 _templateRenderer = templateRenderer;
 }

 public async Task RunAsync()
{
```

```

 var result = await _templateRenderer.RenderAsync(
 "Hello", //the template name
 new HelloModel
 {
 Name = "John"
 }
);
 Console.WriteLine(result);
 }
}
```
* `HelloDemo` is a simple class that injects the `ITemplateRenderer` in its constructor and uses it inside the `RunAsync` method.
* `RenderAsync` gets two fundamental parameters:
  * `templateName`: The name of the template to be rendered (`Hello` in this example).
  * `model`: An object that is used as the `model` inside the template (a `HelloModel` object in this example).

```

The result shown below for this example:

```
```csharp
Hello John :)
```

### ### Anonymous Model

While it is suggested to create model classes for the templates, it would be practical (and possible) to use anonymous objects for simple cases:

```
```csharp
var result = await _templateRenderer.RenderAsync(
    "Hello",
    new
    {
        Name = "John"
    }
);
```

```

In this case, we haven't created a model class, but created an anonymous object as the model.

### ### PascalCase vs snake\_case

PascalCase property names (like `UserName`) is used as snake\_case (like `user\_name`) in the templates.

### ## Localization

It is possible to localize a template content based on the current culture. There are two types of localization options described in the following sections.

### ### Inline localization

```
Inline localization uses the [localization system](Localization.md) to
localize texts inside templates.
```

#### #### Example: Reset Password Link

```
Assuming you need to send an email to a user to reset her/his password. Here,
the template content:
```

```
```
<a title="{${L "ResetMyPasswordTitle"} }%}"
href="${{{model.link}}}">${${L "ResetMyPassword" model.name}}%</a>
````
```

`L` function is used to localize the given key based on the current user culture. You need to define the `ResetMyPassword` key inside your localization file:

```
```json
"ResetMyPasswordTitle": "Reset my password",
"ResetMyPassword": "Hi {0}, Click here to reset your password"
````
```

You also need to declare the localization resource to be used with this template, inside your template definition provider class:

```
```csharp
context.Add(
    new TemplateDefinition(
        "PasswordReset", //Template name
        typeof(DemoResource) //LOCALIZATION RESOURCE
    )
    .WithScribanEngine()
    .WithVirtualFilePath(
        "/Demos/PasswordReset/PasswordReset.tpl", //template content path
        isInlineLocalized: true
    )
);
````
```

That's all. When you render this template like that:

```
```csharp
var result = await _templateRenderer.RenderAsync(
    "PasswordReset", //the template name
    new PasswordResetModel
    {
        Name = "john",
        Link = "https://abp.io/example-link?userId=123&token=ABC"
    }
);
````
```

You will see the localized result:

```
```csharp
<a title="Reset my password" href="https://abp.io/example-
link?userId=123&token=ABC">Hi john, Click here to reset your password</a>
````
```

> If you define the [default localization resource](Localization.md) for your application, then no need to declare the resource type for the template definition.

### ### Multiple Contents Localization

Instead of a single template that uses the localization system to localize the template, you may want to create different template files for each language. It can be needed if the template should be completely different for a specific culture rather than simple text localizations.

#### #### Example: Welcome Email Template

Assuming that you want to send a welcome email to your users, but want to define a completely different template based on the user culture.

First, create a folder and put your templates inside it, like 'en.tpl', 'tr.tpl'... one for each culture you support:

```
![multiple-file-template](images/multiple-file-template.png)
```

Then add your template definition in the template definition provider class:

```
```csharp
context.Add(
    new TemplateDefinition(
        name: "WelcomeEmail",
        defaultCultureName: "en"
    )
    .WithScribanEngine()
    .WithVirtualFilePath(
        "/Demos/WelcomeEmail/Templates", //template content folder
        isInlineLocalized: false
    )
);```

```

* Set **default culture name**, so it fallbacks to the default culture if there is no template for the desired culture.
* Specify **the template folder** rather than a single template file.
* Set 'isInlineLocalized' to 'false' for this case.

That's all, you can render the template for the current culture:

```
```csharp
var result = await _templateRenderer.RenderAsync("WelcomeEmail");
```

```

> Skipped the modal for this example to keep it simple, but you can use models as just explained before.

Specify the Culture

'ITemplateRenderer' service uses the current culture ('CultureInfo.CurrentCulture') if not specified. If you need, you can specify the culture as the 'cultureName' parameter:

```
```csharp
var result = await _templateRenderer.RenderAsync(
 "WelcomeEmail",
 cultureName: "en"
);```

```

## ## Layout Templates

Layout templates are used to create shared layouts among other templates. It is similar to the layout system in the ASP.NET Core MVC / Razor Pages.

### ### Example: Email HTML Layout Template

For example, you may want to create a single layout for all of your email templates.

First, create a template file just like before:

```
```xml
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
</head>
<body>
    {{{{content}}}}
</body>
</html>
```

```

\* A layout template must have a `**{{{{content}}}}**` part as a place holder for the rendered child content.

The register your template in the template definition provider:

```
```csharp
context.Add(
    new TemplateDefinition(
        "EmailLayout",
        isLayout: true //SET isLayout!
    )
    .WithScribanEngine()
    .WithVirtualFilePath(
        "/Demos/EmailLayout/EmailLayout.tpl",
        isInlineLocalized: true
    )
);```

```

Now, you can use this template as the layout of any other template:

```
```csharp
context.Add(
 new TemplateDefinition(
 name: "WelcomeEmail",
 defaultCultureName: "en",
 layout: "EmailLayout" //Set the LAYOUT
)
```

```

```
)  
.WithScribanEngine()  
.WithVirtualFilePath(  
    "/Demos/WelcomeEmail/Templates",  
    isInlineLocalized: false  
)  
);  
};
```

Global Context

ABP passes the `model` that can be used to access to the model inside the template. You can pass more global variables if you need.

An example template content:

```
```  
A global object value: {{myGlobalObject}}
```
```

This template assumes that that is a `myGlobalObject` object in the template rendering context. You can provide it like shown below:

```
```csharp  
var result = await _templateRenderer.RenderAsync(
 "GlobalContextUsage",
 globalContext: new Dictionary<string, object>
 {
 {"myGlobalObject", "TEST VALUE"}
 }
);
};
```

The rendering result will be:

```
```  
A global object value: TEST VALUE  
```
```

## ## Replacing the Existing Templates

It is possible to replace a template defined by a module that used in your application. In this way, you can customize the templates based on your requirements without changing the module code.

### ### Option-1: Using the Virtual File System

The [Virtual File System](Virtual-File-System.md) allows you to override any file by placing the same file into the same path in your project.

#### #### Example: Replace the Standard Email Layout Template

ABP Framework provides an [email sending system](Emailing.md) that internally uses the text templating to render the email content. It defines a standard email layout template in the `/Volo/Abp/Emailing/Templates/Layout.tpl` path. The unique name of the template is `Abp.StandardEmailTemplates.Layout` and this string is defined as a constant on the `Volo.Abp.Emailing.Templates.StandardEmailTemplates` static class.

Do the following steps to replace the template file with your own;

\*\*1)\*\* Add a new file into the same location (`'/Volo/Abp/Emailing/Templates/Layout.tpl'`) in your project:

`![replace-email-layout](images/replace-email-layout.png)`

\*\*2)\*\* Prepare your email layout template:

```
```html
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
</head>
<body>
    <h1>This my header</h1>

    {{{{content}}}}

    <footer>
        This is my footer...
    </footer>
</body>
</html>
```
```

This example simply adds a header and footer to the template and renders the content between them (see the *\*Layout Templates\** section above to understand it).

\*\*3)\*\* Configure the embedded resources in the `'.csproj'` file

- \* Add `[Microsoft.Extensions.FileProviders.Embedded]`(<https://www.nuget.org/packages/Microsoft.Extensions.FileProviders.Embedded>) NuGet package to the project.
- \* Add `<GenerateEmbeddedFilesManifest>true</GenerateEmbeddedFilesManifest>` into the `'<PropertyGroup>...</PropertyGroup>'` section of your `'.csproj'` file.
- \* Add the following code into your `'.csproj'` file:

```
```xml
<ItemGroup>
    <None Remove="Volo\Abp\Emailing\Templates\*.tpl" />
    <EmbeddedResource Include="Volo\Abp\Emailing\Templates\*.tpl" />
</ItemGroup>
```
```

This makes the template files "embedded resource".

\*\*4)\*\* Configure the virtual file system

Configure the `'AbpVirtualFileSystemOptions'` in the `'ConfigureServices'` method of your `[module](Module-Development-Basics.md)` to add the embedded files into the virtual file system:

```
```csharp
Configure<AbpVirtualFileSystemOptions>(options =>
```

```
{  
    options.FileSets.AddEmbedded<BookStoreDomainModule>();  
};  
};
```

'BookStoreDomainModule' should be your module name, in this example code.

> Be sure that your module (directly or indirectly) [depends on](Module-Development-Basics.md) the 'AbpEmailingModule'. Because the VFS can override files based on the dependency order.

Now, your template will be used when you want to render the email layout template.

Option-2: Using the Template Definition Provider

You can create a template definition provider class that gets the email layout template and changes the virtual file path for the template.

Example: Use the '/MyTemplates/EmailLayout.tpl' file instead of the standard template

```
```csharp  
using Volo.Abp.DependencyInjection;
using Volo.Abp.Emailing.Templates;
using Volo.Abp.TextTemplating;

namespace MyProject
{
 public class MyTemplateDefinitionProvider
 : TemplateDefinitionProvider, ITransientDependency
 {
 public override void Define(ITemplateDefinitionContext context)
 {
 var emailLayoutTemplate =
context.GetOrNull(StandardEmailTemplates.Layout);

 emailLayoutTemplate
 .WithVirtualFilePath(
 "/MyTemplates/EmailLayout.tpl",
 isInlineLocalized: true
);
 }
 }
}..
```

You should still add the file '/MyTemplates/EmailLayout.tpl' to the virtual file system as explained before. This approach allows you to locate templates in any folder instead of the folder defined by the depended module.

Beside the template content, you can manipulate the template definition properties, like 'DisplayName', 'Layout' or 'LocalizationSource'.

## ## Advanced Features

This section covers some internals and more advanced usages of the text templating system.

### ### Template Content Provider

`ITemplateRenderer` is used to render the template, which is what you want for most of the cases. However, you can use the `ITemplateContentProvider` to get the raw (not rendered) template contents.

> `ITemplateContentProvider` is internally used by the `ITemplateRenderer` to get the raw template contents.

Example:

```
```csharp
public class TemplateContentDemo : ITransientDependency
{
    private readonly ITemplateContentProvider _templateContentProvider;

    public TemplateContentDemo(ITemplateContentProvider
templateContentProvider)
    {
        _templateContentProvider = templateContentProvider;
    }

    public async Task RunAsync()
    {
        var result = await _templateContentProvider
            .GetContentOrNullAsync("Hello");

        Console.WriteLine(result);
    }
}...```

```

The result will be the raw template content:

```
```
Hello {{model.name}} :)```

```

- \* `GetContentOrNullAsync` returns `null` if no content defined for the requested template.
- \* It can get a `cultureName` parameter that is used if template has different files for different cultures (see Multiple Contents Localization section above).

### ### Template Content Contributor

`ITemplateContentProvider` service uses `ITemplateContentContributor` implementations to find template contents. There is a single pre-implemented content contributor, `VirtualFileTemplateContentContributor`, which gets template contents from the virtual file system as described above.

You can implement the `ITemplateContentContributor` to read raw template contents from another source.

Example:

```
```csharp

```

```

public class MyTemplateContentProvider
    : ITemplateContentContributor, ITransientDependency
{
    public async Task<string>
GetOrNullAsync(TemplateContentContributorContext context)
    {
        var templateName = context.TemplateDefinition.Name;

        //TODO: Try to find content from another source
        return null;
    }
}
...

```

Return `null` if your source can not find the content, so `ITemplateContentProvider` fallbacks to the next contributor.

Template Definition Manager

`ITemplateDefinitionManager` service can be used to get the template definitions (created by the template definition providers).

See Also

- * [The source code of the sample application](<https://github.com/abpframework/abp-samples/tree/master/TextTemplateDemo>) developed and referred through this document.
- * [Localization system](Localization.md).
- * [Virtual File System](Virtual-File-System.md).

7.24 Timing

Timing

Working with times & [time zones](https://en.wikipedia.org/wiki/Time_zone) is always tricky, especially if you need to build a **global system** that is used by users in **different time zones**.

ABP provides a basic infrastructure to make it easy and handle automatically wherever possible. This document covers the ABP Framework services and systems related to time and time zones.

> If you are creating a local application that runs in a single time zone region, you may not need all these systems. But even in this case, it is suggested to use the `IClock` service introduced in this document.

IClock

`DateTime.Now` returns a `DateTime` object with the **local date & time of the server**. A `DateTime` object **doesn't store the time zone information**. So, you can not know the **absolute date & time** stored in this object. You can only make **assumptions**, like assuming that it was created in UTC+05 time zone. The things especially gets complicated when you

save this value to a database and read later, or send it to a client in a ****different time zone****.

One solution to this problem is always use `'DateTime.UtcNow'` and assume all `'DateTime'` objects as UTC time. In this way, you can convert it to the time zone of the target client when needed.

`'IClock'` provides an abstraction while getting the current time, so you can control the kind of the date time (UTC or local) in a single point in your application.

****Example: Getting the current time****

```
```csharp
using Volo.Abp.DependencyInjection;
using Volo.Abp.Timing;

namespace AbpDemo
{
 public class MyService : ITransientDependency
 {
 private readonly IClock _clock;

 public MyService(IClock clock)
 {
 _clock = clock;
 }

 public void Foo()
 {
 //Get the current time!
 var now = _clock.Now;
 }
 }
}...```

```

\* Inject the `'IClock'` service when you need to get the current time. Common base classes (like `ApplicationService`) already injects it and provides as a base property - so, you can directly use as `'Clock'`.

\* Use the `'Now'` property to get the current time.

> Most of the times, `'IClock'` is the only service you need to know and use in your application.

### ### Clock Options

`'AbpClockOptions'` is the [options](Options.md) class that used to set the clock kind.

**\*\*Example: Use UTC Clock\*\***

```
```csharp
Configure<AbpClockOptions>(options =>
{
    options.Kind = DateTimeKind.Utc;
});```

```

Write this inside the `ConfigureServices` method of your [module](Module-Development-Basics.md).

> Default `Kind` is `Unspecified`, that actually make the Clock as it doesn't exists at all. Either make it `Utc` or `Local` if you want to get benefit of the Clock system.

DateTime Normalization

Other important function of the `IClock` is to normalize `DateTime` objects.

Example usage:

```
```csharp
DateTime dateTime = ...; //Get from somewhere
var normalizedDateTime = Clock.Normalize(dateTime)
````
```

`Normalize` method works as described below:

- * Converts the given `DateTime` to the UTC (by using the `DateTime.ToUniversalTime()` method) if current Clock is UTC and given `DateTime` is local.
- * Converts the given `DateTime` to the local (by using the `DateTimeToLocalTime()` method) if current Clock is local and given `DateTime` is UTC.
- * Sets `Kind` of the given `DateTime` (using the `DateTime.SpecifyKind(...)` method) to the `Kind` of the current Clock if given `DateTime`'s `Kind` is `Unspecified`.

`Normalize` method is used by the ABP Framework when the it gets a `DateTime` that is not created by `IClock.Now` and may not be compatible with the current Clock type. Examples;

- * `DateTime` type binding in the ASP.NET Core MVC model binding.
- * Saving data to and reading data from database via [Entity Framework Core](Entity-Framework-Core.md).
- * Working with `DateTime` objects on [JSON deserialization](Json-Serialization.md).

DisableDateTimeNormalization Attribute

`DisableDateTimeNormalization` attribute can be used to disable the normalization operation for desired classes or properties.

Other IClock Properties

In addition to the `Now`, `IClock` service has the following properties:

- * `Kind`: Returns a `DateTimeKind` for the currently used clock type (`DateTimeKind.Utc`, `DateTimeKind.Local` or `DateTimeKind.Unspecified`).
- * `SupportsMultipleTimezone`: Returns `true` if currently used clock is UTC.

Time Zones

This section covers the ABP Framework infrastructure related to managing time zones.

TimeZone Setting

ABP Framework defines **a setting**, named '`Abp.Timing.Timezone`', that can be used to set and get the time zone for a user, [`tenant`](Multi-Tenancy.md) or globally for the application. The default value is '`UTC`'.

See the [`setting documentation`](Settings.md) to learn more about the setting system.

ITimezoneProvider

`'ITimezoneProvider'` is a service to simple convert [`Windows Time Zone Id`](https://support.microsoft.com/en-us/help/973627/microsoft-time-zone-index-values) values to [`Iana Time Zone Name`](https://www.iana.org/time-zones) values and vice versa. It also provides methods to get list of these time zones and get a '`TimeZoneInfo`' with a given name.

It has been implemented using the [`TimeZoneConverter`](https://github.com/mj1856/TimeZoneConverter) library.

7.25 Virtual File System

Virtual File System

The Virtual File System makes it possible to manage files that do not physically exist on the file system (disk). It's mainly used to embed (js, css, image..) files into assemblies and use them like physical files at runtime.

Installation

> Most of the times you don't need to manually install this package since it comes pre-installed with the [`application startup template`](Startup-Templates/Application.md).

[`Volo.Abp.VirtualFileSystem`](https://www.nuget.org/packages/Volo.Abp.VirtualFileSystem) is the main package of the Virtual File System.

Use the ABP CLI to add this package to your project:

- * Install the [`ABP CLI`](https://docs.abp.io/en/abp/latest/CLI), if you haven't installed it.
- * Open a command line (terminal) in the directory of the '`.csproj`' file you want to add the '`Volo.Abp.VirtualFileSystem`' package.
- * Run '`abp add-package Volo.Abp.VirtualFileSystem`' command.

If you want to do it manually, install the [`Volo.Abp.VirtualFileSystem`](https://www.nuget.org/packages/Volo.Abp.VirtualFileSystem) NuGet package to your project and add '`[DependsOn(typeof(AbpVirtualFileSystemModule))]`' to the [`ABP module`](Module-Development-Basics.md) class inside your project.

Working with the Embedded Files

Embedding the Files

A file should be first marked as **embedded resource** to embed the file into the assembly. The easiest way to do it is to select the file from the **Solution Explorer** and set **Build Action** to **Embedded Resource** from the **Properties** window. Example:

```
![build-action-embedded-resource-sample](images/build-action-embedded-resource-sample.png)
```

If you want to add multiple files, this can be tedious. Alternatively, you can directly edit your `*.csproj` file:

```
```C#
<ItemGroup>
 <EmbeddedResource Include="MyResources***.*" />
 <Content Remove="MyResources***.*" />
</ItemGroup>
````
```

This configuration recursively adds all files under the **MyResources** folder of the project (including the files you will add in the future).

Embedding a file in the project/assembly may cause problems if a file name contains some special chars. To overcome this limitation;

1. Add [\[Microsoft.Extensions.FileProviders.Embedded\]](https://www.nuget.org/packages/Microsoft.Extensions.FileProviders.Embedded)(<https://www.nuget.org/packages/Microsoft.Extensions.FileProviders.Embedded>) NuGet package to the project that contains the embedded resource(s).
2. Add `<GenerateEmbeddedFilesManifest>true</GenerateEmbeddedFilesManifest>` into the `<PropertyGroup>...</PropertyGroup>` section of your `*.csproj` file.

> While these two steps are optional and ABP can work without these configuration, it is strongly suggested to make it.

Configure the AbpVirtualFileSystemOptions

Use `AbpVirtualFileSystemOptions` [options class](Options.md) to register the embedded files to the virtual file system in the `ConfigureServices` method of your [module](Module-Development-Basics.md).

Example: Add embedded files to the virtual file system

```
```csharp
Configure<AbpVirtualFileSystemOptions>(options =>
{
 options.FileSets.AddEmbedded<MyModule>();
});
````
```

The `AddEmbedded` extension method takes a class, finds all embedded files from the **assembly** of the given class** and registers them to the virtual file system.

`AddEmbedded` can get two optional parameters;

```
* `baseNamespace`: This may only needed if you didn't configure the
`GenerateEmbeddedFileManifest` step explained above and your root namespace
is not empty. In this case, set your root namespace here.
* `baseFolder`: If you don't want to expose all embedded files in the
project, but only want to expose a specific folder (and sub folders/files),
then you can set the base folder relative to your project root folder.
```

Example: Add files under the 'MyResources' folder in the project

```
```csharp
Configure<AbpVirtualFileSystemOptions>(options =>
{
 options.FileSets.AddEmbedded<MyModule>(
 baseNamespace: "Acme.BookStore",
 baseFolder: "/MyResources"
);
})```
```

```

This example assumes;

- * Your project root (default) namespace is 'Acme.BookStore'.
- * Your project has a folder, named 'MyResources'
- * You only want to add 'MyResources' folder to the virtual file system.

IVirtualFileProvider

After embedding a file into an assembly and registering it to the virtual file system, the 'IVirtualFileProvider' interface can be used to get the file or directory contents:

```
```C#
public class MyService : ITransientDependency
{
 private readonly IVirtualFileProvider _virtualFileProvider;

 public MyService(IVirtualFileProvider virtualFileProvider)
 {
 _virtualFileProvider = virtualFileProvider;
 }

 public void Test()
 {
 //Getting a single file
 var file = _virtualFileProvider
 .GetFileInfo("/MyResources/js/test.js");

 var fileContent = file.ReadAllText();

 //Getting all files/directories under a directory
 var directoryContents = _virtualFileProvider
 .GetDirectoryContents("/MyResources/js");
 }
}
```
```
ASP.NET Core Integration

```

The Virtual File System is well integrated to ASP.NET Core:

- \* Virtual files can be used just like physical (static) files in a web application.
- \* Js, css, image files and all other web content types can be embedded into assemblies and used just like the physical files.
- \* An application (or another module) can \*\*override a virtual file\*\* of a module just like placing a file with the same name and extension into the same folder of the virtual file.

### ### Static Virtual File Folders

By default, ASP.NET Core only allows the `wwwroot` folder to contain the static files consumed by the clients. When you use the virtual File System, the following folders also can contain static files:

- \* Pages
- \* Views
- \* Components
- \* Themes

This allows to add `\*.js`, `\*.css`... files near to your `\*.cshtml` file that is easier to develop and maintain your project.

### ## Dealing With Embedded Files During Development

Embedding a file into an assembly and being able to use it from another project just by referencing the assembly (or adding a NuGet package) is invaluable for creating a re-usable module. However, it makes it a little bit harder to develop the module itself.

Let's assume that you're developing a module that contains an embedded JavaScript file. Whenever you change this file you must re-compile the project, re-start the application and refresh the browser page to take the change. Obviously, this is very time consuming and tedious.

What is needed is the ability for the application to directly use the physical file at development time and a browser refresh reflects any change made in the JavaScript file. The `ReplaceEmbeddedByPhysical` method makes all this possible.

The example below shows an application that depends on a module (`MyModule`) that contains embedded files. The application can access to the source code of the module at development time.

```
```C#
[DependsOn(typeof(MyModule))]
public class MyWebAppModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        var hostingEnvironment = context.Services.GetHostingEnvironment();

        if (hostingEnvironment.IsDevelopment()) //only for development time
        {
            Configure<AbpVirtualFileSystemOptions>(options =>
            {

```

```

        options.FileSets.ReplaceEmbeddedByPhysical<MyModule>(
            Path.Combine(
                hostingEnvironment.ContentRootPath,
                string.Format(
                    "..{0}MyModuleProject",
                    Path.DirectorySeparatorChar
                )
            )
        );
    }
}
```

```

The code above assumes that `MyWeb AppModule` and `MyModule` are two different projects in a Visual Studio solution and `MyWeb AppModule` depends on the `MyModule`.

> The [application startup template](Startup-Templates/Application.md) already uses this technique for the localization files. So, when you change a localization file it automatically detects the change.

### [## Replacing/Overriding Virtual Files](#)

Virtual File System creates a unified file system on runtime, where the actual files are distributed into different modules in the development time.

If two modules adds a file to the same virtual path (like `my-path/my-file.css`), the one added later overrides/replaces the previous one ([module dependency](Module-Development-Basics.md) order determines the order of the files being added).

This feature allows your application to override/replace any virtual file defined a module that is used by your application. This is one of the fundamental extensibility features of the ABP Framework.

So, if you need to replace a file of a module, just create the file in the exactly same path in your module/application

### [### Physical Files](#)

Physical files always override the virtual files. That means if you put a file under the `/wwwroot/my-folder/my-file.css`, it will override the file in the same location of the virtual file system. So, you need to know the file paths defined in the modules to override them.

## 8 Architecture

### 8.1 Modularity

#### 8.1.1 Basics

##### [# Modularity](#)

##### [## Introduction](#)

ABP Framework was designed to support to build fully modular applications and systems where every module may have entities, services, database integration, APIs, UI components and so on;

- \* This document introduces the basics of the module system.
- \* [\[Module development best practice guide\]](#)(Best-Practices/Index.md) explains some \*\*best practices\*\* to develop \*\*re-usable application modules\*\* based on \*\*DDD\*\* principles and layers. A module designed based on this guide will be \*\*database independent\*\* and can be deployed as a \*\*microservice\*\* if needed.
- \* [\[Pre-built application modules\]](#)(Modules/Index.md) are \*\*ready to use\*\* in any kind of application.
- \* [\[Module startup template\]](#)(Startup-Templates/Module.md) is a jump start way to \*\*create a new module\*\*.
- \* [\[ABP CLI\]](#)(CLI.md) has commands to support modular development.
- \* All other framework features are compatible to the modularity system.

## ## Module Class

Every module should define a module class. The simplest way of defining a module class is to create a class derived from `'`AbpModule`'` as shown below:

```
```C#
public class BlogModule : AbpModule
{
}

````
```

## ### Configuring Dependency Injection & Other Modules

### #### ConfigureServices Method

`'`ConfigureServices`'` is the main method to add your services to the dependency injection system and configure other modules. Example:

> These methods have Async versions too, and if you want to make asynchronous calls inside these methods, override the asynchronous versions instead of the synchronous ones.

```
```C#
public class BlogModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext
context)
    {
        //...
    }
}

````
```

You can register dependencies one by one as stated in Microsoft's [\[documentation\]](#)(<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection>). But ABP has a \*\*conventional dependency registration system\*\* which automatically register all services in your assembly. See the [\[dependency Injection\]](#)(Dependency-Injection.md) documentation for more about the dependency injection system.

You can also configure other services and modules in this way. Example:

```
```C#
public class BlogModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        //Configure default connection string for the application
        Configure<AbpDbConnectionOptions>(options =>
        {
            options.ConnectionStrings.Default = ".....";
        });
    }
}...```

```

> `ConfigureServices` method has an asynchronous version too: `ConfigureServicesAsync`. If you want to make asynchronous calls (use the `await` keyword) inside this method, override the asynchronous version instead of the synchronous one. If you override both asynchronous and synchronous versions, only the asynchronous version will be executed.

See the [\[Configuration\]\(Configuration.md\)](#) document for more about the configuration system.

Pre & Post Configure Services

``AbpModule`` class also defines ``PreConfigureServices`` and ``PostConfigureServices`` methods to override and write your code just before and just after ``ConfigureServices``. Notice that the code you have written into these methods will be executed before/after the ``ConfigureServices`` methods of all other modules.

> These methods have asynchronous versions too. If you want to make asynchronous calls inside these methods, override the asynchronous versions instead of the synchronous ones.

Application Initialization

Once all the services of all modules are configured, the application starts by initializing all modules. In this phase, you can resolve services from ``IServiceProvider`` since it's ready and available.

OnApplicationInitialization Method

You can override ``OnApplicationInitialization`` method to execute code while application is being started.

Example:

```
```C#
public class BlogModule : AbpModule
{
 public override void OnApplicationInitialization(
 ApplicationInitializationContext context)
 {
 var myService = context.ServiceProvider.GetService<MyService>();```

```

```
 myService.DoSomething();
 }
}
```

```

`'OnApplicationInitialization'` method has an asynchronous version too. If you want to make asynchronous calls (use the `'await'` keyword) inside this method, override the asynchronous version instead of the synchronous one.

****Example:****

```
```csharp
public class BlogModule : AbpModule
{
 public override Task OnApplicationInitializationAsync(
 ApplicationInitializationContext context)
 {
 var myService = context.ServiceProvider.GetService<MyService>();
 await myService.DoSomethingAsync();
 }
}
```

```

> If you override both asynchronous and synchronous versions, only the asynchronous version will be executed.

`'OnApplicationInitialization'` is generally used by the startup module to construct the middleware pipeline for ASP.NET Core applications.

****Example:****

```
```C#
[DependsOn(typeof(AbpAspNetCoreMvcModule))]
public class AppModule : AbpModule
{
 public override void
OnApplicationInitialization(ApplicationInitializationContext context)
 {
 var app = context.GetApplicationBuilder();
 var env = context.GetEnvironment();

 if (env.IsDevelopment())
 {
 app.UseDeveloperExceptionPage();
 }

 app.UseMvcWithDefaultRoute();
 }
}
```

```

You can also perform startup logic if your module requires it

Pre & Post Application Initialization

`'AbpModule'` class also defines `'OnPreApplicationInitialization'` and `'OnPostApplicationInitialization'` methods to override and write your code just before and just after `'OnApplicationInitialization'`. Notice that the

code you have written into these methods will be executed before/after the ``OnApplicationInitialization`` methods of all other modules.

> These methods have asynchronous versions too, and if you want to make asynchronous calls inside these methods, override the asynchronous versions instead of the synchronous ones.

Application Shutdown

Lastly, you can override ``OnApplicationShutdown`` method if you want to execute some code while application is being shutdown.

> This methods has asynchronous version too. If you want to make asynchronous calls inside this method, override the asynchronous version instead of the synchronous one.

Module Dependencies

In a modular application, it's not unusual for one module to depend upon another module(s). An ABP module must declare a ``[DependsOn]`` attribute if it does have a dependency upon another module, as shown below:

```
```C#
[DependsOn(typeof(AbpAspNetCoreMvcModule))]
[DependsOn(typeof(AbpAutofacModule))]
public class BlogModule
{
 //...
}
...```

```

You can use multiple ``DependsOn`` attribute or pass multiple module types to a single ``DependsOn`` attribute depending on your preference.

A depended module may depend on another module, but you only need to define your direct dependencies. ABP investigates the dependency graph for the application at startup and initializes/shutdowns modules in the correct order.

### ## Additional Module Assemblies

ABP automatically registers all the services of your module to the [dependency injection](Dependency-Injection.md) system. It finds the service types by scanning types in the assembly that defines your module class. That assembly is considered as the main assembly of your module.

Typically, every assembly contains a separate module class definition. Then modules depend on each other using the `DependsOn` attribute as explained in the previous section. However, in some rare cases, your module may consist of multiple assemblies, and only one of them defines a module class, and you want to make the other assemblies parts of your module. In that case, you can use the `AdditionalAssembly` attribute as shown below:

```
```csharp
[DependsOn(...)] // Your module dependencies as you normally do
[AdditionalAssembly(typeof(BlogService))] // A type in the target assembly
public class BlogModule
{```

```

```
//...
}...
```

In this example, we assume that the `BlogService` class is inside one assembly (``csproj``) and the `BlogModule` class is inside another assembly (`csproj`). With the `AdditionalAssembly` definition, ABP will load the assembly containing the `BlogService` class as a part of the blog module.

Notice that `BlogService` is only an arbitrary selected type in the target assembly. It is just used to indicate the related assembly. You could use any type in the assembly.

> WARNING: If you need to use the `AdditionalAssembly`, be sure that you don't design your system in a wrong way. With this example above, the `BlogService` class' assembly should normally have its own module class and the `BlogModule` should depend on it using the `DependsOn` attribute. Do not use the `AdditionalAssembly` attribute when you can already use the `DependsOn` attribute.

Framework Modules vs Application Modules

There are **two types of modules.** They don't have any structural difference but categorized by functionality and purpose:

- **Framework modules**: These are **core modules of the framework** like caching, emailing, theming, security, serialization, validation, EF Core integration, MongoDB integration... etc. They do not have application/business functionalities but makes your daily development easier by providing common infrastructure, integration and abstractions.
- **Application modules**: These modules implement **specific application/business functionalities** like blogging, document management, identity management, tenant management... etc. They generally have their own entities, services, APIs and UI components. See [pre-built application modules](Modules/Index.md).

8.1.2 Plug-In Modules

Plug-In Modules

It is possible to load [modules](Module-Development-Basics.md) as plug-ins. That means you may not reference to a module's assembly in your solution, but you can load that module in the application startup just like any other module.

Basic Usage

`IServiceCollection.AddApplication<T>()` extension method can get options to configure the plug-in sources.

Example: Load plugins from a folder

```
```csharp
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Volo.Abp.Modularity.Plugins;
```

```

namespace MyPlugInDemo.Web
{
 public class Startup
 {
 public void ConfigureServices(IServiceCollection services)
 {
 services.AddApplication<MyPlugInDemoWebModule>(options =>
 {
 options.PlugInSources.AddFolder(@"D:\Temp\MyPlugIns");
 });
 }

 public void Configure(IApplicationBuilder app)
 {
 app.InitializeApplication();
 }
 }
}
...

```

\* This is the `Startup` class of a typical ASP.NET Core application.  
\* `PlugInSources.AddFolder` gets a folder path and to load assemblies  
(typically `dll`s) in that folder.

That's all. ABP will discover the modules in the given folder, configure and initialize them just like regular modules.

### ### Plug-In Sources

`options.PlugInSources` is actually a list of `IPlugInSource` implementations and `AddFolder` is just a shortcut for the following expression:

```
```csharp
options.PlugInSources.Add(new FolderPlugInSource(@"D:\Temp\MyPlugIns"));
```

```

> `AddFolder()` only looks for the assembly file in the given folder, but not looks for the sub-folders. You can pass `SearchOption.AllDirectories` as a second parameter to explore plug-ins also from the sub-folders, recursively.

There are two more built-in Plug-In Source implementations:

- \* `PlugInSources.AddFiles()` gets a list of assembly (typically `dll`) files. This is a shortcut of using `FilePlugInSource` class.
- \* `PlugInSources.AddTypes()` gets a list of module class types. If you use this, you need to load the assemblies of the modules yourself, but it provides flexibility when needed. This is a shortcut of using `TypePlugInSource` class.

If you need, you can create your own `IPlugInSource` implementation and add to the `options.PlugInSources` just like the others.

### ## Example: Creating a Simple Plug-In

Create a simple \*\*Class Library Project\*\* in a solution:

![simple-plugin-library](images/simple-plugin-library.png)

You can add the ABP Framework packages that you need to use in the module. At least, you should add the '[`Volo.Abp.Core`](#)' package to the project, Execute the following command in the folder of the .csproj file that you want to install the package on:

```
```bash
abp add-package Volo.Abp.Core
````
```

If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [[the package description page](#)](<https://abp.io/package-detail/Volo.Abp.Core>).

Every [[module](#)](Module-Development-Basics.md) must declare a class derived from the '[`AbpModule`](#)'. Here, a simple module class that resolves a service and initializes it on the application startup:

```
```csharp
using Microsoft.Extensions.DependencyInjection;
using Volo.Abp;
using Volo.Abp.Modularity;

namespace MyPlugIn
{
    public class MyPlungInModule : AbpModule
    {
        public override void
OnApplicationInitialization(ApplicationInitializationContext context)
        {
            var myService = context.ServiceProvider
                .GetRequiredService<MyService>();

            myService.Initialize();
        }
    }
}
````
```

'[`MyService`](#)' can be any class registered to [[Dependency Injection](#)](Dependency-Injection.md) system, as show below:

```
```csharp
using Microsoft.Extensions.Logging;
using Volo.Abp.DependencyInjection;

namespace MyPlugIn
{
    public class MyService : ITransientDependency
    {
        private readonly ILogger<MyService> _logger;

        public MyService(ILogger<MyService> logger)
        {
            _logger = logger;
        }

        public void Initialize()
````
```

```
 {
 _logger.LogInformation("MyService has been initialized");
 }
 }
}
```

Build the project, open the build folder, find the `MyPlugIn.dll`:

![simple-plug-in-dll-file](images/simple-plug-in-dll-file.png)

Copy `MyPlugIn.dll` into the plug-in folder (`D:\Temp\MyPlugIns` for this example).

> Please delete the `MyPlugIn.deps.json` file if you use `build folder` folder as `PlugInSources`.

If you have configured the main application like described above (see Basic Usage section), you should see the `MyService has been initialized` log in the application startup.

## ## Example: Creating a Plug-In With Razor Pages

Creating plug-ins with views inside requires a bit more attention.

> This example assumes you've [created a new web application](https://abp.io/get-started) using the application startup template and MVC / Razor Pages UI.

Create a new \*\*Class Library\*\* project in a solution:

![simple-razor-plugin](images/simple-razor-plugin.png)

Edit the `\*.csproj` file content:

```
```xml
<Project Sdk="Microsoft.NET.Sdk.Web">

    <PropertyGroup>
        <TargetFramework>net5.0</TargetFramework>
        <OutputType>Library</OutputType>
        <IsPackable>true</IsPackable>
    </PropertyGroup>

    <ItemGroup>
        <PackageReference Include="Volo.Abp.AspNetCore.Mvc.UI.Theme.Shared"
Version="4.0.1" />
    </ItemGroup>
</Project>
```

* Changed `Sdk` to `Microsoft.NET.Sdk.Web`.

* Added `OutputType` and `IsPackable` properties.

* Added `Volo.Abp.AspNetCore.Mvc.UI.Theme.Shared` NuGet package.

> Depending on

[Volo.Abp.AspNetCore.Mvc.UI.Theme.Shared](https://www.nuget.org/packages/Volo

.Abp.AspNetCore.Mvc.UI.Theme.Shared) package is not required. You can reference to a more base package like [Volo.Abp.AspNetCore.Mvc](<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc/>). However, if you will build a UI page/component, it is suggested to reference to the [Volo.Abp.AspNetCore.Mvc.UI.Theme.Shared](<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.UI.Theme.Shared>) package since it is the most high-level package without depending on a particular [theme](UI/AspNetCore/Theming.md). If there is no problem to depend on a particular theme, you can directly reference to the theme's package to be able to use the theme-specific features in your plug-in.

Then create your module class in the plug-in:

```
```csharp
using System.IO;
using System.Reflection;
using Microsoft.AspNetCore.Mvc.ApplicationParts;
using Microsoft.Extensions.DependencyInjection;
using Volo.Abp.AspNetCore.Mvc.UI.Theme.Shared;
using Volo.Abp.Modularity;

namespace MyMvcUIPlugIn
{
 [DependsOn(typeof(AbpAspNetCoreMvcUiThemeSharedModule))]
 public class MyMvcUIPlugInModule : AbpModule
 {
 public override void PreConfigureServices(ServiceConfigurationContext context)
 {
 PreConfigure<IMvcBuilder>(mvcBuilder =>
 {
 //Add plugin assembly
 mvcBuilder.PartManager.ApplicationParts.Add(new
AssemblyPart(typeof(MyMvcUIPlugInModule).Assembly));

 //Add CompiledRazorAssemblyPart if the PlugIn module contains
//razor views.
 mvcBuilder.PartManager.ApplicationParts.Add(new
CompiledRazorAssemblyPart(typeof(MyMvcUIPlugInModule).Assembly));
 });
 }
 }
}...```

```

- \* Depending on the `AbpAspNetCoreMvcUiThemeSharedModule` since we added the related NuGet package.
- \* Adding the plug-in's assembly as `AssemblyPart` and `CompiledRazorAssemblyPart` to the `PartManager` of ASP.NET Core MVC. This is required by ASP.NET Core. Otherwise, your controllers or views inside the plug-in doesn't work.

You can now add a razor page, like `MyPlugInPage.cshtml` inside the `Pages` folder:

```
```html
@page
```

```
@model MyMvcUIPlugIn.Pages.MyPlugInPage
<h1>Welcome to my plug-in page</h1>
<p>This page is located inside a plug-in module! :)</p>
` ``
```

Now, you can build the plug-in project. It will produce the following output:

```
![simple-razor-plug-in-dll-file](images/simple-razor-plug-in-dll-file.png)
```

Copy the `MyMvcUIPlugIn.dll` into the plug-in folder (`D:\Temp\MyPlugIns` for this example).

If you have configured the main application like described above (see Basic Usage section), you should be able to visit the `/MyPlugInPage` URL when your application:

```
![simple-plugin-output](images/simple-plugin-output.png)
```

Discussions

In real world, your plug-in may have some external dependencies. Also, your application might be designed to support plug-ins. All these are your own system requirements. What ABP does is just loading modules on the application startup. What you do inside that modules is up to you.

However, we can provide a few suggestions for some common cases.

Library Dependencies

For package/dll dependencies, you can copy the related dlls to the plug-in folder. ABP automatically loads all assemblies in the folder and your plug-in will work as expected.

> See [Microsoft's documentation](<https://docs.microsoft.com/en-us/dotnet/core/tutorials/creating-app-with-plugin-support#plugin-with-library-dependencies>) for some additional explanations for that case.

Database Schema

If your module uses a relational database and [Entity Framework Core](Entity-Framework-Core.md), it will need to have its tables available in the database. There are different ways to ensure the tables have been created when an application uses the plug-in. Some examples;

1. The Plugin may check if the database tables does exists and create the tables on the application startup or migrate them if the plug-in has been updated and requires some schema changes. You can use EF Core's migration API to do that.
2. You can improve the `DbMigrator` application to find migrations of the plug-ins and execute them.

There may be other solutions. For example, if your DB admin doesn't allow you to change the database schema in the application code, you may need to manually send a SQL file to the database admin to apply it to the database.

8.1.3 Best Practices

8.1.3.1 Overall

Module Development Best Practices & Conventions

Introduction

This document describes the **best practices** and **conventions** for those who want to develop **modules** that satisfy the following specifications:

- * Develop modules that conform to the **Domain Driven Design** patterns & best practices.
- * Develop modules with **DBMS and ORM independence**.
- * Develop modules that can be used as a **remote service / microservice** as well as being compatible with a **monolithic** application.

Also, this guide is mostly usable for general **application development**.

Guides

- * Overall
 - * [Module Architecture](Module-Architecture.md)
- * Domain Layer
 - * [Entities](Entities.md)
 - * [Repositories](Repositories.md)
 - * [Domain Services](Domain-Services.md)
- * Application Layer
 - * [Application Services](Application-Services.md)
 - * [Data Transfer Objects](Data-Transfer-Objects.md)
- * Data Access
 - * [Entity Framework Core Integration](Entity-Framework-Core-Integration.md)
 - * [MongoDB Integration](MongoDB-Integration.md)

See Also

- * [E-Book: Implementing Domain Driven Design](<https://abp.io/books/implementing-domain-driven-design>)

8.1.3.2 Module Architecture

Module Architecture Best Practices & Conventions

Solution Structure

- * **Do** create a separated Visual Studio solution for every module.
- * **Do** name the solution as **CompanyName.ModuleName** (for core ABP modules, it's **Volo.Abp.ModuleName**).
- * **Do** develop the module as layered, so it has several packages (projects) those are related to each other.
 - * Every package has its own module definition file and explicitly declares the dependencies for the depended packages/modules.

Layers & Packages

The following diagram shows the packages of a well-layered module and dependencies of those packages between them:

![module-layers-and-packages](../images/module-layers-and-packages.jpg)

The ultimate goal is to allow an application to use the module in a flexible manner. Example applications:

- * **A)** A **monolithic** application;
 - * Adds references to the **Web** and the **Application** packages.
 - * Adds a reference to one of the **EF Core** or the **MongoDB** packages based on the preference.
 - * The result;
 - * The application **can** show **UI** of the module.
 - * It hosts the **application** and **domain** layers in the **same process** (that's why it needs to have a reference to a database integration package).
 - * This application also **serves** the module's **HTTP API** (since it includes the **HttpApi** package through the **Web** package).
- * **B)** An application that just serves the module as a **microservice**;
 - * Adds a reference to **HttpApi** and **Application** packages.
 - * Adds a reference to one of the **EF Core** or the **MongoDB** packages based on the preference.
 - * The result;
 - * The application **can** not show **UI** of the module since it does not have a reference to the **Web** package.
 - * It hosts the **application** and **domain** layers in the **same process** (that's why it needs to have a reference to a database integration package).
 - * This application **serves** the module's **HTTP API** (as the main goal of the application).
- * **C)** An application that shows the module **UI** but does not host the application (just uses it as a remote service that is hosted by the application A or B);
 - * Adds a reference to the **Web** and the **HttpApi.Client** packages.
 - * Configures the remote endpoint for the **HttpApi.Client** package.
 - * The result;
 - * The application **can** show **UI** of the module.
 - * It does not host the application and domain layers of the module in the same process. Instead, uses it as a **remote service**.
 - * This application also **serves** the module's **HTTP API** (since it includes the **HttpApi** package through the **Web** package).
- * **D)** A **client** application (or **microservice**) that just uses the module as a remote service (that is hosted by the application A, B or C);
 - * Adds a reference to the **HttpApi.Client** package.
 - * Configures the remote endpoint for the **HttpApi.Client** package.
 - * The result;
 - * The application can use all the functionality of the module as a **remote client**.
 - * The application is just a client and **can** not **serve** the **HTTP API** of the module.
 - * The application is just a client and **can** not **show** the **UI** of the module.
- * **E)** A proxy application that hosts the **HTTP API** of the module but just forwards all requests to another application (that is hosted by the application A, B or C);
 - * Adds a reference to the **HttpApi** and **HttpApi.Client** packages.
 - * Configures the remote endpoint for the **HttpApi.Client** package.

```
* The result;  
  * The application can use all the functionality of the module as a  
**remote client**.  
  * This application also **serves** the module's **HTTP API**, but  
actually works just like a proxy by redirecting all requests (for the module)  
to another remote server.
```

Next section describes the packages in more details.

Domain Layer

```
* **Do** divide the domain layer into two projects:  
  * **Domain.Shared** package, named as  
*CompanyName.ModuleName.Domain.Shared*, that contains constants, enums and  
other types those can be safely shared with the all layers of the module.  
This package can also be shared to 3rd-party clients. It can not contain  
entities, repositories, domain services or any other business objects.  
  * **Domain** package, named as *CompanyName.ModuleName.Domain*, that  
contains entities, repository interfaces, domain service interfaces and their  
implementations and other domain objects.  
  * Domain package depends on the **Domain.Shared** package.
```

Application Layer

```
* **Do** divide the application layer into two projects:  
  * **Application.Contracts** package, named as  
*CompanyName.ModuleName.Application.Contracts*, that contains application  
service interfaces and related data transfer objects.  
    * Application contract package depends on the **Domain.Shared** package.  
  * **Application** package, named as *CompanyName.ModuleName.Application*,  
that contains application service implementations.  
    * Application package depends on the **Domain** and the  
**Application.Contracts** packages.
```

Infrastructure Layer

```
* **Do** create a separated integration package for each ORM/database  
integration like Entity Framework Core and MongoDB.  
  * **Do**, for instance, create a  
*CompanyName.ModuleName.EntityFrameworkCore* package that abstracts the  
Entity Framework Core integration. ORM integration packages depend on the  
**Domain** package.  
  * **Do not** depend on other layers from the ORM/database integration  
package.  
* **Do** create a separated integration package for each major library that  
is planned to be replaceable by another library without effecting the other  
packages.
```

HTTP Layer

```
* **Do** create an **HTTP API** package, named as  
*CompanyName.ModuleName.HttpApi*, to develop a REST style HTTP API for the  
module.  
  * HTTP API package only depends on the **Application.Contracts** package.  
It does not depend on the Application package.  
  * **Do** create a Controller for each application service (generally by  
implementing their interfaces). These controllers uses the application
```

```
service interfaces to delegate the actions. It just configures routes, HTTP methods and other web related stuffs if needed.  
* **Do** create an **HTTP API Client** package, named as *CompanyName.ModuleName.HttpApi.Client*, to provide client services for the HTTP API package. Those client services implement application interfaces as clients to a remote endpoint.  
  * HTTP API Client package only depends on the **Application.Contracts** package.  
  * **Do** use dynamic HTTP C# client proxy feature of the ABP framework.
```

Web Layer

```
* **Do** create a **Web** package, named as *CompanyName.ModuleName.Web*, that contains pages, views, scripts, styles, images and other UI components.  
  * Web package only depends on the **HttpApi** package.
```

8.1.3.3 Domain Layer

8.1.3.3.1 Entities

Entity Best Practices & Conventions

Entities

Every aggregate root is also an entity. So, these rules are valid for aggregate roots too unless aggregate root rules override them.

- **Do** define entities in the **domain layer**.

Primary Constructor

```
* **Do** define a **primary constructor** that ensures the validity of the entity on creation. Primary constructors are used to create a new instance of the entity by the application code.
```

- **Do** define primary constructor as `'public'`, `'internal'` or `'protected internal'` based on the requirements. If it's not public, the entity is expected to be created by a domain service.
- **Do** always initialize sub collections in the primary constructor.
- **Do not** generate `'Guid'` keys inside the constructor. Get it as a parameter, so the calling code will use `'IGuidGenerator'` to generate a new `'Guid'` value.

Parameterless Constructor

- **Do** always define a `'protected'` parameterless constructor to be compatible with ORMs.

References

```
- **Do** always **reference** to other aggregate roots **by Id**. Never add navigation properties to other aggregate roots.
```

Other Class Members

- **Do** always define properties and methods as `'virtual'` (except `'private'` methods, obviously). Because some ORMs and dynamic proxy tools require it.

- **Do** keep the entity as always **valid** and **consistent** within its own boundary.
 - **Do** define properties with `private`, `protected`, `internal` or `protected internal` setter where it is needed to protect the entity consistency and validity.
 - **Do** define `public`, `internal` or `protected internal` (virtual) **methods** to change the properties (with non-public setters) if necessary.
 - **Do** return the entity object (`this`) from the setter methods.

Aggregate Roots

Primary Keys

- * **Do** always use a **Id** property for the aggregate root key.
- * **Do not** use **composite keys** for aggregate roots.
- * **Do** use **Guid** as the **primary key** of all aggregate roots.

Base Class

- * **Do** inherit from the `AggregateRoot< TKey >` or one of the audited classes (`CreationAuditedAggregateRoot< TKey >`, `AuditedAggregateRoot< TKey >` or `FullAuditedAggregateRoot< TKey >`) based on requirements.

Aggregate Boundary

- * **Do** keep aggregates **as small as possible**. Most of the aggregates will only have primitive properties and will not have sub collections. Consider these as design decisions:
 - * **Performance** & **memory** cost of loading & saving aggregates (keep in mind that an aggregate is normally loaded & saved as a single unit). Larger aggregates will consume more CPU & memory.
 - * **Consistency** & **validity** boundary.

Example

Aggregate Root

```
```C#
public class Issue : FullAuditedAggregateRoot< Guid > //Using Guid as the
key/identifier
{
 public virtual string Title { get; private set; } //Changed using the
SetTitle() method
 public virtual string Text { get; set; } //Can be directly changed. null
values are allowed
 public virtual Guid? MilestoneId { get; set; } //Reference to another
aggregate root
 public virtual bool IsClosed { get; private set; }
 public virtual IssueCloseReason? CloseReason { get; private set; } //Just
an enum type
 public virtual Collection< IssueLabel > Labels { get; protected set; }
//Sub collection

 protected Issue()
 {
 /* This constructor is for ORMs to be used while getting the entity
from database.
 * - No need to initialize the Labels collection
 }
}
```

```

 since it will be overrided from the database.
 - It's protected since proxying and deserialization tools
 may not work with private constructors.
 */
}

//Primary constructor
public Issue(
 Guid id, //Get Guid value from the calling code
 [NotNull] string title, //Indicate that the title can not be null.
 string text = null,
 Guid? milestoneId = null) //Optional argument
{
 Id = id;
 Title = Check.NotNullOrWhiteSpace(title, nameof(title)); //Validate
 Text = text;
 MilestoneId = milestoneId;

 Labels = new Collection<IssueLabel>(); //Always initialize the
 collection
}

public virtual Issue SetTitle([NotNull] string title)
{
 Title = Check.NotNullOrWhiteSpace(title, nameof(title)); //Validate
 return this;
}

/* AddLabel & RemoveLabel methods manages the Labels collection
 * in a safe way (prevents adding the same label twice) */

public virtual Issue AddLabel(Guid labelId)
{
 if (Labels.Any(l => l.LabelId == labelId))
 {
 return;
 }

 Labels.Add(new IssueLabel(Id, labelId));
 return this;
}

public virtual Issue RemoveLabel(Guid labelId)
{
 Labels.RemoveAll(l => l.LabelId == labelId);
 return this;
}

/* Close & ReOpen methods protect the consistency
 * of the IsClosed and the CloseReason properties. */

public virtual void Close(IssueCloseReason reason)
{
 IsClosed = true;
 CloseReason = reason;
}

public virtual void ReOpen()

```

```

 {
 IsClosed = false;
 CloseReason = null;
 }
 }...
}

The Entity

```C#
public class IssueLabel : Entity
{
    public virtual Guid IssueId { get; private set; }
    public virtual Guid LabelId { get; private set; }

    protected IssueLabel()
    {

    }

    public IssueLabel(Guid issueId, Guid labelId)
    {
        IssueId = issueId;
        LabelId = labelId;
    }
}...

```

References

- * Effective Aggregate Design by Vaughn Vernon
http://dddcommunity.org/library/vernon_2011

8.1.3.3.2 Repositories

Repository Best Practices & Conventions

Repository Interfaces

- * **Do** define repository interfaces in the **domain layer**.
- * **Do** define a repository interface (like `IIdentityUserRepository`) and create its corresponding implementations for **each aggregate root**.
 - * **Do** always use the created repository interface from the application code.
 - * **Do not** use generic repository interfaces (like ` IRepository<IdentityUser, Guid>`) from the application code.
 - * **Do not** use ` IQueryable< TEntity >` features in the application code (domain, application... layers).

For the example aggregate root:

```

```C#
public class IdentityUser : AggregateRoot<Guid>
{
 //...
}...

```

Define the repository interface as below:

```
```C#
public interface IIdentityUserRepository : IBasicRepository<IdentityUser,
Guid>
{
    //...
}
```
* **Do not** inherit the repository interface from the ` IRepository< TEntity, TKey >` interface. Because it inherits the ` IQueryable` and the repository should not expose ` IQueryable` to the application.
* **Do** inherit the repository interface from ` IBasicRepository< TEntity, TKey >` (as normally) or a lower-featured interface, like ` IReadOnlyRepository< TEntity, TKey >` (if it's needed).
* **Do not** define repositories for entities those are **not aggregate roots**.
```

### ### Repository Methods

\* \*\*Do\*\* define all repository methods as \*\*asynchronous\*\*.  
\* \*\*Do\*\* add an \*\*optional\*\* ` cancellationToken` parameter to every method of the repository. Example:

```
```C#
Task<IdentityUser> FindByNormalizedUserNameAsync(
    [NotNull] string normalizedUserName,
    CancellationToken cancellationToken = default
);
```

```

\* \*\*Do\*\* add an optional ` bool includeDetails = true` parameter (default value is ` true`) for every repository method which returns a \*\*single entity\*\*. Example:

```
```C#
Task<IdentityUser> FindByNormalizedUserNameAsync(
    [NotNull] string normalizedUserName,
    bool includeDetails = true,
    CancellationToken cancellationToken = default
);
```

```

This parameter will be implemented for ORMs to eager load sub collections of the entity.

\* \*\*Do\*\* add an optional ` bool includeDetails = false` parameter (default value is ` false`) for every repository method which returns a \*\*list of entities\*\*. Example:

```
```C#
Task<List<IdentityUser>> GetListByNormalizedRoleNameAsync(
    string normalizedRoleName,
    bool includeDetails = false,
    CancellationToken cancellationToken = default
);
```

```

....

- \* \*\*Do not\*\* create composite classes to combine entities to get from repository with a single method call. Examples: `*UserWithRoles*`, `*UserWithTokens*`, `*UserWithRolesAndTokens*`. Instead, properly use `'includeDetails'` option to add all details of the entity when needed.
- \* \*\*Avoid\*\* to create projection classes for entities to get less property of an entity from the repository. Example: Avoid to create `BasicUserView` class to select a few properties needed for the use case needs. Instead, directly use the aggregate root class. However, there may be some exceptions for this rule, where:
  - \* Performance is so critical for the use case and getting the whole aggregate root highly impacts the performance.

#### ### See Also

- \* [Entity Framework Core Integration](Entity-Framework-Core-Integration.md)
- \* [MongoDB Integration](MongoDB-Integration.md)

##### 8.1.3.3.3 Domain Services

#### ## Domain Services Best Practices & Conventions

> \*\*This document is not ready yet. Please see the [Domain Services](../Domain-Services.md) document.\*\*

##### 8.1.3.4 Application Layer

###### 8.1.3.4.1 Application Services

#### ## Application Services Best Practices & Conventions

- \* \*\*Do\*\* create an application service for each \*\*aggregate root\*\*.

#### ### Application Service Interface

- \* \*\*Do\*\* define an `'interface'` for each application service in the `**application contracts**` package.
- \* \*\*Do\*\* inherit from the `'IApplicationService'` interface.
- \* \*\*Do\*\* use the `'AppService'` postfix for the interface name (ex: `'IProductAppService'`).
- \* \*\*Do\*\* create DTOs (Data Transfer Objects) for inputs and outputs of the service.
- \* \*\*Do not\*\* get/return entities for the service methods.
- \* \*\*Do\*\* define DTOs based on the [DTO best practices](Data-Transfer-Objects.md).

#### #### Outputs

- \* \*\*Avoid\*\* to define too many output DTOs for same or related entities. Instead, define a `**basic**` and a `**detailed**` DTO for an entity.

#### ##### Basic DTO

**\*\*Do\*\*** define a **basic** DTO for an aggregate root.

- Include all the **primitive properties** directly on the aggregate root.
  - Exception: Can **exclude** properties for **security** reasons (like `'User.Password'`).
- Include all the **sub collections** of the entity where every item in the collection is a simple **relation DTO**.
- Inherit from one of the **extensible entity DTO** classes for aggregate roots (and entities implement the `'IHasExtraProperties'`).

Example:

```
```c#
[Serializable]
public class IssueDto : ExtensibleFullAuditedEntityDto<Guid>
{
    public string Title { get; set; }
    public string Text { get; set; }
    public Guid? MilestoneId { get; set; }
    public Collection<IssueLabelDto> Labels { get; set; }
}

[Serializable]
public class IssueLabelDto
{
    public Guid IssueId { get; set; }
    public Guid LabelId { get; set; }
}
```

```

#### ##### Detailed DTO

**\*\*Do\*\*** define a **detailed** DTO for an entity if it has reference(s) to other aggregate roots.

- \* Include all the **primitive properties** directly on the entity.
  - Exception-1: Can **exclude** properties for **security** reasons (like `'User.Password'`).
  - Exception-2: **Do** exclude reference properties (like `'MilestoneId'` in the example above). Will already add details for the reference properties.
- \* Include a **basic** DTO property for every reference property.
- \* Include all the **sub collections** of the entity where every item in the collection is the **basic** DTO of the related entity.

Example:

```
```C#
[Serializable]
public class IssueWithDetailsDto : ExtensibleFullAuditedEntityDto<Guid>
{
    public string Title { get; set; }
    public string Text { get; set; }
    public MilestoneDto Milestone { get; set; }
    public Collection<LabelDto> Labels { get; set; }
}

[Serializable]

```

```

public class MilestoneDto : ExtensibleEntityDto<Guid>
{
    public string Name { get; set; }
    public bool IsClosed { get; set; }
}

[Serializable]
public class LabelDto : ExtensibleEntityDto<Guid>
{
    public string Name { get; set; }
    public string Color { get; set; }
}
...

```

Inputs

- * **Do not** define any property in an input DTO that is not used in the service class.
- * **Do not** share input DTOs between application service methods.
- * **Do not** inherit an input DTO class from another one.
 - * **May** inherit from an abstract base DTO class and share some properties between different DTOs in that way. However, should be very careful in that case because manipulating the base DTO would effect all related DTOs and service methods. Avoid from that as a good practice.

Methods

- * **Do** define service methods as asynchronous with **Async** postfix.
- * **Do not** repeat the entity name in the method names.
 - * Example: Define `GetAsync(...)` instead of `GetProductAsync(...)` in the `IProductAppService`.

Getting A Single Entity

- * **Do** use the `GetAsync` **method name**.
- * **Do** get Id with a **primitive** method parameter.
- * Return the **detailed DTO**. Example:

```

```C#
Task<QuestionWithDetailsDto> GetAsync(Guid id);
```

```

Getting A List Of Entities

- * **Do** use the `GetListAsync` **method name**.
- * **Do** get a single DTO argument for **filtering**, **sorting** and **paging** if necessary.
 - * **Do** implement filters optional where possible.
 - * **Do** implement sorting & paging properties as optional and provide default values.
 - * **Do** limit maximum page size (for performance reasons).
- * **Do** return a list of **detailed DTO**s. Example:

```

```C#
Task<List<QuestionWithDetailsDto>> GetListAsync(QuestionListQueryDto
queryDto);
```

```

Creating A New Entity

- * **Do** use the `CreateAsync` **method name**.
- * **Do** get a **specialized input** DTO to create the entity.
- * **Do** inherit the DTO class from the `ExtensibleObject` (or any other class implements the `IHasExtraProperties`) to allow to pass extra properties if needed.
- * **Do** use **data annotations** for input validation.
 - * Share constants between domain wherever possible (via constants defined in the **domain shared** package).
- * **Do** return **the detailed** DTO for new created entity.
- * **Do** only require the **minimum** info to create the entity but provide possibility to set others as optional properties.

Example **method**:

```
```C#
Task<QuestionWithDetailsDto> CreateAsync(CreateQuestionDto questionDto);
````
```

The related **DTO**:

```
```C#
[Serializable]
public class CreateQuestionDto : ExtensibleObject
{
 [Required]
 [StringLength(QuestionConsts.MaxTitleLength,
 MinimumLength = QuestionConsts.MinTitleLength)]
 public string Title { get; set; }

 [StringLength(QuestionConsts.MaxTextLength)]
 public string Text { get; set; } //Optional

 public Guid? CategoryId { get; set; } //Optional
}
````
```

Updating An Existing Entity

- **Do** use the `UpdateAsync` **method name**.
- **Do** get a **specialized input** DTO to update the entity.
- **Do** inherit the DTO class from the `ExtensibleObject` (or any other class implements the `IHasExtraProperties`) to allow to pass extra properties if needed.
- **Do** get the Id of the entity as a separated primitive parameter. Do not include to the update DTO.
- **Do** use **data annotations** for input validation.
 - Share constants between domain wherever possible (via constants defined in the **domain shared** package).
- **Do** return **the detailed** DTO for the updated entity.

Example:

```
```C#
Task<QuestionWithDetailsDto> UpdateAsync(Guid id, UpdateQuestionDto
updateQuestionDto);
````
```

Deleting An Existing Entity

- **Do** use the `DeleteAsync` method name.
- **Do** get Id with a primitive method parameter. Example:

```
```C#
Task DeleteAsync(Guid id);
````
```

Other Methods

- * **Can** define additional methods to perform operations on the entity. Example:

```
```C#
Task<int> VoteAsync(Guid id, VoteType type);
````
```

This method votes a question and returns the current score of the question.

Application Service Implementation

- * **Do** develop the application layer completely independent from the web layer.
- * **Do** implement application service interfaces in the application layer.
 - * **Do** use the naming convention. Ex: Create `ProductAppService` class for the `IProductAppService` interface.
 - * **Do** inherit from the `ApplicationService` base class.
- * **Do** make all public methods virtual, so developers may inherit and override them.
- * **Do not** make private methods. Instead make them protected virtual, so developers may inherit and override them.

Using Repositories

- * **Do** use the specifically designed repositories (like `IProductRepository`).
- * **Do not** use generic repositories (like ` IRepository<Product>`).

Querying Data

- * **Do not** use LINQ/SQL for querying data from database inside the application service methods. It's repository's responsibility to perform LINQ/SQL queries from the data source.

Extra Properties

- * **Do** use either `MapExtraPropertiesTo` extension method ([see](./Object-Extensions.md)) or configure the object mapper (`MapExtraProperties`) to allow application developers to be able to extend the objects and services.

Manipulating / Deleting Entities

- * **Do** always get all the related entities from repositories to perform the operations on them.

* **Do** call repository's Update/UpdateAsync method after updating an entity. Because, not all database APIs support change tracking & auto update.

Handle files

* **Do not** use any web components like 'IFormFile' or 'Stream' in the application services. If you want to serve a file you can use 'byte[]'.
* **Do** use a 'Controller' to handle file uploading then pass the 'byte[]' of the file to the application service method.

Using Other Application Services

* **Do not** use other application services of the same module/application. Instead;
 * Use domain layer to perform the required task.
 * Extract a new class and share between the application services to accomplish the code reuse when necessary. But be careful to don't couple two use cases. They may seem similar at the beginning, but may evolve to different directions by time. So, use code sharing carefully.
* **Can** use application services of others only if;
 * They are parts of another module / microservice.
 * The current module has only reference to the application contracts of the used module.

8.1.3.4.2 Data Transfer Objects

Data Transfer Objects Best Practices & Conventions

* **Do** define DTOs in the **application contracts** package.
* **Do** inherit from the pre-built **base DTO classes** where possible and necessary (like 'EntityDto<TKey>', 'CreationAuditedEntityDto<TKey>', 'AuditedEntityDto<TKey>', 'FullAuditedEntityDto<TKey>' and so on).
 * **Do** inherit from the **extensible DTO** classes for the **aggregate roots** (like 'ExtensibleAuditedEntityDto<TKey>'), because aggregate roots are extensible objects and extra properties are mapped to DTOs in this way.
* **Do** define DTO members with **public getter and setter**.
* **Do** use **data annotations** for **validation** on the properties of DTOs those are inputs of the service.
* **Do** not add any **logic** into DTOs except implementing 'IValidatableObject' when necessary.
* **Do** mark all DTOs as **[Serializable]** since they are already serializable and developers may want to binary serialize them.

8.1.3.5 Data Access

8.1.3.5.1 Entity Framework Core Integration

Entity Framework Core Integration Best Practices

> See [Entity Framework Core Integration document](../Entity-Framework-Core.md) for the basics of the EF Core integration.

- **Do** define a separated `DbContext` interface and class for each module.
- **Do not** rely on lazy loading on the application development.
- **Do not** enable lazy loading for the `DbContext`.

DbContext Interface

- **Do** define an **interface** for the `DbContext` that inherits from `IEfCoreDbContext`.
- **Do** add a `ConnectionStringName` **attribute** to the `DbContext` interface.
- **Do** add `DbSet< TEntity >` **properties** to the `DbContext` interface for only aggregate roots. Example:

```
```C#
[ConnectionStringName("AbpIdentity")]
public interface IIdentityDbContext : IEfCoreDbContext
{
 DbSet<IdentityUser> Users { get; }
 DbSet<IdentityRole> Roles { get; }
}
```

```

* **Do not** define `set;` for the properties in this interface.

DbContext class

- * **Do** inherit the `DbContext` from the `AbpDbContext< TDbContext >` class.
 - * **Do** add a `ConnectionStringName` attribute to the `DbContext` class.
 - * **Do** implement the corresponding `interface` for the `DbContext` class.
- Example:

```
```C#
[ConnectionStringName("AbpIdentity")]
public class IdentityDbContext : AbpDbContext<IdentityDbContext>,
IIdentityDbContext
{
 public DbSet<IdentityUser> Users { get; set; }
 public DbSet<IdentityRole> Roles { get; set; }

 public IdentityDbContext(DbContextOptions<IdentityDbContext> options)
 : base(options)
 {
 }

 //code omitted for brevity
}
```

```

Table Prefix and Schema

- **Do** add static `TablePrefix` and `Schema` **properties** to the `DbContext` class. Set default value from a constant. Example:

```
```C#
public static string TablePrefix { get; set; } =
AbpIdentityConsts.DefaultDbTablePrefix;
```

```

```
public static string Schema { get; set; } =
AbpIdentityConsts.DefaultDbSchema;
````
```

- \*\*Do\*\* always use a short 'TablePrefix' value for a module to create \*\*unique table names\*\* in a shared database. 'Abp' table prefix is reserved for ABP core modules.
- \*\*Do\*\* set 'Schema' to 'null' as default.

### ### Model Mapping

- \*\*Do\*\* explicitly \*\*configure all entities\*\* by overriding the 'OnModelCreating' method of the 'DbContext'. Example:

```
```C#
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);
    builder.ConfigureIdentity();
}
````
```

- \*\*Do not\*\* configure model directly in the 'OnModelCreating' method. Instead, create an \*\*extension method\*\* for 'ModelBuilder'. Use Configure\*ModuleName\* as the method name. Example:

```
```C#
public static class IdentityDbContextModelBuilderExtensions
{
    public static void ConfigureIdentity([NotNull] this ModelBuilder builder)
    {
        Check.NotNull(builder, nameof(builder));

        builder.Entity<IdentityUser>(b =>
        {
            b.ToTable(AbpIdentityDbProperties.DbTablePrefix + "Users",
AbpIdentityDbProperties.DbSchema);
            b.ConfigureByConvention();
            //code omitted for brevity
        });

        builder.Entity<IdentityUserClaim>(b =>
        {
            b.ToTable(AbpIdentityDbProperties.DbTablePrefix + "UserClaims",
AbpIdentityDbProperties.DbSchema);
            b.ConfigureByConvention();
            //code omitted for brevity
        });

        //code omitted for brevity
    }
}
````
```

\* \*\*Do\*\* call 'b.ConfigureByConvention();' for each entity mapping (as shown above).

### ### Repository Implementation

- \*\*Do\*\* \*\*inherit\*\* the repository from the `EfCoreRepository<TDbContext, TEntity, TKey>` class and implement the corresponding repository interface.  
Example:

```
```C#
public class EfCoreIdentityUserRepository
    : EfCoreRepository<IIdentityDbContext, IdentityUser, Guid>,
    IIdentityUserRepository
{
    public EfCoreIdentityUserRepository(
        IDbContextProvider<IIdentityDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }
}
````
```

\* \*\*Do\*\* use the `DbContext` interface as the generic parameter, not the class.  
\* \*\*Do\*\* pass the `cancellationToken` to EF Core using the `GetCancellationToken` helper method. Example:

```
```C#
public virtual async Task<IdentityUser> FindByNormalizedUserNameAsync(
    string normalizedUserName,
    bool includeDetails = true,
    CancellationToken cancellationToken = default)
{
    return await (await GetDbSetAsync())
        .IncludeDetails(includeDetails)
        .FirstOrDefaultAsync(
            u => u.NormalizedUserName == normalizedUserName,
            GetCancellationToken(cancellationToken))
    );
}
````
```

`GetCancellationToken` fallbacks to the `ICancellationTokenProvider.Token` to obtain the cancellation token if it is not provided by the caller code.

- \*\*Do\*\* create a `IncludeDetails` \*\*extension method\*\* for the `IQueryable<TEntity>` for each aggregate root which has \*\*sub collections\*\*.  
Example:

```
```C#
public static IQueryable<IdentityUser> IncludeDetails(
    this IQueryable<IdentityUser> queryable,
    bool include = true)
{
    if (!include)
    {
        return queryable;
    }

    return queryable
        .Include(x => x.Roles)
        .Include(x => x.Logins)
}
```

```
        .Include(x => x.Claims)
        .Include(x => x.Tokens);
    }
}
```

* **Do** use the `IncludeDetails` extension method in the repository methods just like used in the example code above (see `FindByNormalizedUserNameAsync`).

- **Do** override `WithDetails` method of the repository for aggregates root which have **sub collections**. Example:

```
```C#
public override async Task<IQueryable<IdentityUser>> WithDetailsAsync()
{
 // Uses the extension method defined above
 return await GetQueryableAsync().IncludeDetails();
}
}
```

### ### Module Class

- \*\*Do\*\* define a module class for the Entity Framework Core integration package.

- \*\*Do\*\* add `DbContext` to the `IServiceCollection` using the `AddAbpDbContext<TDbContext>` method.

- \*\*Do\*\* add implemented repositories to the options for the `AddAbpDbContext<TDbContext>` method. Example:

```
```C#
[DependsOn(
    typeof(AbpIdentityDomainModule),
    typeof(AbpEntityFrameworkCoreModule)
)]
public class AbpIdentityEntityFrameworkCoreModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        context.Services.AddAbpDbContext<IdentityDbContext>(options =>
        {
            options.AddRepository<IdentityUser,
EfCoreIdentityUserRepository>();
            options.AddRepository<IdentityRole,
EfCoreIdentityRoleRepository>();
        });
    }
}
```

8.1.3.5.2 MongoDB Integration

MongoDB Integration

* Do define a separated `MongoDbContext` interface and class for each module.

```
### MongoDBContext Interface

- **Do** define an **interface** for the `MongoDbContext` that inherits from `IAbpMongoDbContext`.
- **Do** add a `ConnectionStringName` **attribute** to the `MongoDbContext` interface.
- **Do** add `IMongoCollection< TEntity >` **properties** to the `MongoDbContext` interface only for the aggregate roots. Example:
```

```
```C#
[ConnectionStringName("AbpIdentity")]
public interface IAbpIdentityMongoDbContext : IAbpMongoDbContext
{
 IMongoCollection<IdentityUser> Users { get; }
 IMongoCollection<IdentityRole> Roles { get; }
}
```

```

MongoDBContext class

```
- **Do** inherit the `MongoDbContext` from the `AbpMongoDbContext` class.
- **Do** add a `ConnectionStringName` attribute to the `MongoDbContext` class.
- **Do** implement the corresponding `interface` for the `MongoDbContext` class. Example:
```

```
```c#
[ConnectionStringName("AbpIdentity")]
public class AbpIdentityMongoDbContext : AbpMongoDbContext,
IAbpIdentityMongoDbContext
{
 public IMongoCollection<IdentityUser> Users =>
Collection<IdentityUser>();
 public IMongoCollection<IdentityRole> Roles =>
Collection<IdentityRole>();

 //code omitted for brevity
}
```

```

Collection Prefix

```
- **Do** add static `CollectionPrefix` **property** to the `DbContext` class. Set default value from a constant. Example:
```

```
```c#
public static string CollectionPrefix { get; set; } =
AbpIdentityConsts.DefaultDbTablePrefix;
```

```

Used the same constant defined for the EF Core integration table prefix in this example.

```
- **Do** always use a short `CollectionPrefix` value for a module to create **unique collection names** in a shared database. `Abp` collection prefix is reserved for ABP core modules.
```

Collection Mapping

- **Do** explicitly **configure all aggregate roots** by overriding the `CreateModel` method of the `MongoDbContext`. Example:

```
```c#
protected override void CreateModel(IMongoModelBuilder modelBuilder)
{
 base.CreateModel(modelBuilder);

 modelBuilder.ConfigureIdentity();
}..
```

- \*\*Do not\*\* configure model directly in the `CreateModel` method. Instead, create an \*\*extension method\*\* for the `IMongoModelBuilder`. Use `Configure\*ModuleName\*` as the method name. Example:

```
```c#
public static class AbpIdentityMongoDbContextExtensions
{
    public static void ConfigureIdentity(
        this IMongoModelBuilder builder,
        Action<IdentityMongoModelBuilderConfigurationOptions> optionsAction =
null)
    {
        Check.NotNull(builder, nameof(builder));

        builder.Entity<IdentityUser>(b =>
        {
            b.CollectionName = AbpIdentityDbProperties.DbTablePrefix +
"Users";
        });

        builder.Entity<IdentityRole>(b =>
        {
            b.CollectionName = AbpIdentityDbProperties.DbTablePrefix +
"Roles";
        });
    }
}..
```

Repository Implementation

- **Do** **inherit** the repository from the `MongoDBRepository<TMongoDbContext, TEntity, TKey>` class and implement the corresponding repository interface. Example:

```
```c#
public class MongoIdentityUserRepository
 : MongoDBRepository<IAbpIdentityMongoDbContext, IdentityUser, Guid>,
 IIdentityUserRepository
{
 public MongoIdentityUserRepository(
 IMongoDbContextProvider<IAbpIdentityMongoDbContext>
dbContextProvider)
 : base(dbContextProvider)
 {
```

```

 }
 } ..

- **Do** pass the `cancellationToken` to the MongoDB Driver using the
`GetCancellationToken` helper method. Example:

```c#
public async Task<IdentityUser> FindByNormalizedUserNameAsync(
    string normalizedUserName,
    bool includeDetails = true,
    CancellationToken cancellationToken = default)
{
    return await (await GetMongoQueryableAsync())
        .FirstOrDefaultAsync(
            u => u.NormalizedUserName == normalizedUserName,
            GetCancellationToken(cancellationToken)
        );
} ..
```

```

`GetCancellationToken` fallbacks to the `ICancellationProvider.Token` to obtain the cancellation token if it is not provided by the caller code.

- \* \*\*Do\*\* ignore the `includeDetails` parameters for the repository implementation since MongoDB loads the aggregate root as a whole (including sub collections) by default.
- \* \*\*Do\*\* use the `GetMongoQueryableAsync()` method to obtain an `IQueryable< TEntity >` to perform queries wherever possible. Because;
  - \* `GetMongoQueryableAsync()` method automatically uses the `ApplyDataFilters` method to filter the data based on the current data filters (like soft delete and multi-tenancy).
  - \* Using `IQueryable< TEntity >` makes the code as much as similar to the EF Core repository implementation and easy to write and read.
- \* \*\*Do\*\* implement data filtering if it is not possible to use the `GetMongoQueryable()` method.

### ### Module Class

- \*\*Do\*\* define a module class for the MongoDB integration package.
- \*\*Do\*\* add `MongoDbContext` to the `IServiceCollection` using the `AddMongoDbContext< TMongoDbContext >` method.
- \*\*Do\*\* add implemented repositories to the options for the `AddMongoDbContext< TMongoDbContext >` method. Example:

```

```c#
[DependsOn(
    typeof(AbpIdentityDomainModule),
    typeof(AbpUsersMongoDbModule)
)]
public class AbpIdentityMongoDbModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext
context)
    {
        context.Services.AddMongoDbContext<AbpIdentityMongoDbContext>(options
=>
{
```

```

```

 options.AddRepository<IdentityUser,
MongoIdentityUserRepository>();
 options.AddRepository<IdentityRole,
MongoIdentityRoleRepository>();
 });
}
}..

```

Notice that this module class also calls the static `BsonClassMap` configuration method defined above.

## 8.1.4 Customizing/Extending Modules

### 8.1.4.1 Overall

#### # Customizing the Existing Modules

ABP Framework has been designed to support to build fully [modular applications](Module-Development-Basics.md) and systems. It also provides some [pre-built application modules](Modules/Index.md) those are \*\*ready to use\*\* in any kind of application.

For example, you can \*\*re-use\*\* the [Identity Management Module](Modules/Identity.md) to add user, role and permission management to your application. The [application startup template](Startup-Templates/Application.md) already comes with Identity and some other modules \*\*pre-installed\*\*.

#### ## Re-Using an Application Module

You have two options to re-use an application module.

##### ### As Package References

You can add \*\*NuGet\*\* & \*\*NPM\*\* package references of the related module to your application and configure the module (based on its documentation) to integrate to your application.

As mentioned before, the [application startup template](Startup-Templates/Application.md) already comes with some \*\*fundamental modules pre-installed\*\*. It uses the modules as NuGet & NPM package references.

This approach has the following benefits:

- \* Your solution will be \*\*clean\*\* and only contains your \*\*own application code\*\*.
- \* You can \*\*easily upgrade\*\* a module when a new version is available. `abp update` [CLI](CLI.md) command makes it even easier. In this way, you can continue to get \*\*new features and bug fixes\*\*.

However, there is a drawback:

- \* You may not able to \*\*customize\*\* the module because the module source is not in your solution.

This document explains \*\*how to customize or extend\*\* a depended module without need to change its source code. While it is limited compared to a full source code change opportunity, there are still some good ways to make some customizations.

If you don't think to make huge changes on the pre-built modules, re-using them as package reference is the recommended way.

### ### Including the Source Code

If you want to make \*\*huge changes\*\* or add \*\*major features\*\* on a pre-built module, but the available extension points are not enough, you can consider to directly work the source code of the depended module.

In this case, you typically \*\*add the source code\*\* of the module to your solution and replace \*\*every\*\* package reference in the solution with its corresponding local project references. \*\*[ABP CLI](CLI.md)\*\*'s `add-module` command automates this process for you with the `--with-source-code` parameter. This command can also replace a module by its source code if the module already installed as NuGet packages.

### #### Separating the Module Solution

You may prefer to not include the module source code \*\*directly into your solution\*\*. Every module consists of 10+ project files and adding \*\*multiple modules\*\* may impact on the \*\*size\*\* of your solution \*\*load & development time\*\*. Also, you may have different development teams working on different modules, so you don't want to make the module code available to the application development team.

In any case, you can create a \*\*separate solution\*\* for the desired module and depend on the module as project references out of the solution. We do it like that for the [abp repository](<https://github.com/abpframework/abp/>).

> One problem we see is Visual Studio doesn't play nice with this kind of approach (it doesn't support well to have references to local projects out of the solution directory). If you get error while building the application (depends on an external module), run `dotnet restore` in the command line after opening the application's solution in the Visual Studio.

### #### Publishing the Customized Module as Packages

One alternative scenario could be re-packaging the module source code (as NuGet/NPM packages) and using as package references. You can use a local private NuGet/NPM server for your company, for example.

## ## Module Customization / Extending Approaches

This section suggests some approaches if you decided to use pre-built application modules as NuGet/NPM package references. The following documents explain how to customize/extend existing modules in different ways.

### ### Module Entity Extension System

> Module entity extension system is the \*\*main and high level extension system\*\* that allows you to \*\*define new properties\*\* for existing entities

of the depended modules. It automatically \*\*adds properties to the entity, database, HTTP API and the user interface\*\* in a single point.

See the [\[Module Entity Extensions document\]](#)(Module-Entity-Extensions.md) to learn how to use it.

### ### Extending Entities

If you only need to get/set extra data on an existing entity, follow the [\[Extending Entities\]](#)(Customizing-Application-Modules-Extending-Entities.md) document.

### ### Overriding Services/Components

In addition to the extensibility systems, you can partially or completely override any service or user interface page/component.

- \* [\[Overriding Services\]](#)(Customizing-Application-Modules-Overriding-Services.md)
- \* [\[Overriding the User Interface\]](#)(Customizing-Application-Modules-Overriding-User-Interface.md)

### ### Additional UI Extensibility Points

There are some low level systems that you can control entity actions, table columns and page toolbar of a page defined by a module.

#### #### Entity Actions

Entity action extension system allows you to add a new action to the action menu for an entity on the user interface;

- \* [\[Entity Action Extensions for ASP.NET Core UI\]](#)(UI/AspNetCore/Entity-Action-Extensions.md)
- \* [\[Entity Action Extensions for Blazor UI\]](#)(UI/Blazor/Entity-Action-Extensions.md)
- \* [\[Entity Action Extensions for Angular\]](#)(UI/Angular/Entity-Action-Extensions.md)

#### #### Data Table Column Extensions

Data table column extension system allows you to add a new column in the data table on the user interface;

- \* [\[Data Table Column Extensions for ASP.NET Core UI\]](#)(UI/AspNetCore/Data-Table-Column-Extensions.md)
- \* [\[Data Table Column Extensions for Blazor UI\]](#)(UI/Blazor/Data-Table-Column-Extensions.md)
- \* [\[Data Table Column Extensions for Angular\]](#)(UI/Angular/Data-Table-Column-Extensions.md)

#### #### Page Toolbar

Page toolbar system allows you to add components to the toolbar of a page;

- \* [\[Page Toolbar Extensions for ASP.NET Core UI\]](#)(UI/AspNetCore/Page-Toolbar-Extensions.md)

- \* [Page Toolbar Extensions for Blazor UI](UI/Blazor/Page-Toolbar-Extensions.md)
- \* [Page Toolbar Extensions for Angular](UI/Angular/Page-Toolbar-Extensions.md)

#### #### Others

- \* [Dynamic Form Extensions for Angular](UI/Angular/Dynamic-Form-Extensions.md)

#### ## See Also

Also, see the following documents:

- \* See [the localization document](Localization.md) to learn how to extend existing localization resources.
- \* See [the settings document](Settings.md) to learn how to change setting definitions of a depended module.
- \* See [the authorization document](Authorization.md) to learn how to change permission definitions of a depended module.

#### 8.1.4.2 Module Entity Extension System

##### # Module Entity Extensions

Module entity extension system is a \*\*high level\*\* extension system that allows you to \*\*define new properties\*\* for existing entities of the depended modules. It automatically \*\*adds properties to the entity, database, HTTP API and the user interface\*\* in a single point.

> The module must be developed the \*Module Entity Extensions\* system in mind. All the \*\*official modules\*\* supports this system wherever possible.

##### ## Quick Example

Open the `YourProjectNameModuleExtensionConfigurator` class inside the `Domain.Shared` project of your solution and change the `ConfigureExtraProperties` method as shown below to add a `SocialSecurityNumber` property to the `IdentityUser` entity of the [Identity Module](Modules/Identity.md).

```
```csharp
public static void ConfigureExtraProperties()
{
    OneTimeRunner.Run(() =>
    {
        ObjectExtensionManager.Instance.Modules()
            .ConfigureIdentity(identity =>
        {
            identity.ConfigureUser(user =>
            {
                user.AddOrUpdateProperty<string>() //property type: string
                    "SocialSecurityNumber", //property name
                    property =>
                {
                    //validation rules
                }
            });
        });
    });
}
```

```

        property.Attributes.Add(new RequiredAttribute());
        property.Attributes.Add(
            new StringLengthAttribute(64) {
                MinimumLength = 4
            }
        );
    }

    //...other configurations for this property
}
);
);
);
}
...

```

>This method is called inside the `YourProjectNameDomainSharedModule` at the beginning of the application. `OneTimeRunner` is a utility class that guarantees to execute this code only one time per application, since multiple calls are unnecessary.

- * `ObjectExtensionManager.Instance.Modules()` is the starting point to configure a module. `ConfigureIdentity(...)` method is used to configure the entities of the Identity Module.
 - * `identity.ConfigureUser(...)` is used to configure the user entity of the identity module. Not all entities are designed to be extensible (since it is not needed). Use the intellisense to discover the extensible modules and entities.
 - * `user.AddOrUpdateProperty<string>(...)` is used to add a new property to the user entity with the `string` type (`AddOrUpdateProperty` method can be called multiple times for the same property of the same entity. Each call can configure the options of the same property, but only one property is added to the entity with the same property name). You can call this method with different property names to add more properties.
 - * `SocialSecurityNumber` is the name of the new property.
 - * `AddOrUpdateProperty` gets a second argument (the `property =>` lambda expression) to configure additional options for the new property.
- * We can add data annotation attributes like shown here, just like adding a data annotation attribute to a class property.

Create & Update Forms

Once you define a property, it appears in the create and update forms of the related entity:

![add-new-property-to-user-form](images/add-new-property-to-user-form.png)

`SocialSecurityNumber` field comes into the form. Next sections will explain the localization and the validation for this new property.

Data Table

New properties also appear in the data table of the related page:

![add-new-property-to-user-form](images/add-new-property-to-user-table.png)

`SocialSecurityNumber` column comes into the table. Next sections will explain the option to hide this column from the data table.

Property Options

There are some options that you can configure while defining a new property.

Display Name

You probably want to set a different (human readable) display name for the property that is shown on the user interface.

Don't Want to Localize?

If your application is not localized, you can directly set the `DisplayName` for the property to a `FixedLocalizableString` object. Example:

```
```csharp
property =>
{
 property.DisplayName = new FixedLocalizableString("Social security no");
}
````
```

Localizing the Display Name

If you want to localize the display name, you have two options.

Localize by Convention

Instead of setting the `property.DisplayName`, you can directly open your localization file (like `en.json`) and add the following entry to the `texts` section:

```
```json
"SocialSecurityNumber": "Social security no"
````
```

Define the same `SocialSecurityNumber` key (the property name you've defined before) in your localization file for each language you support. That's all!

In some cases, the localization key may conflict with other keys in your localization files. In such cases, you can use the `DisplayName:` prefix for display names in the localization file (`DisplayName:SocialSecurityNumber` as the localization key for this example). Extension system looks for prefixed version first, then fallbacks to the non prefixed name (it then fallbacks to the property name if you haven't localized it).

> This approach is recommended since it is simple and suitable for most scenarios.

Localize using the `DisplayName` Property

If you want to specify the localization key or the localization resource, you can still set the `DisplayName` option:

```
```csharp
property =>
{
 property.DisplayName =

```

```
 LocalizableString.Create<MyProjectNameResource>(
 "UserSocialSecurityNumberDisplayName"
);
}...
* `MyProjectNameResource` is the localization resource and
`UserSocialSecurityNumberDisplayName` is the localization key in the
localization resource.

> See [the localization document](Localization.md) if you want to learn more
about the localization system.
```

#### #### Default Value

A default value is automatically set for the new property, which is the natural default value for the property type, like `null` for `string`, `false` for `bool` or `0` for `int`.

There are two ways to override the default value:

#### ##### DefaultValue Option

`DefaultValue` option can be set to any value:

```
```csharp
property =>
{
    property.DefaultValue = 42;
}...
```

```

#### ##### DefaultValueFactory Options

`DefaultValueFactory` can be set to a function that returns the default value:

```
```csharp
property =>
{
    property.DefaultValueFactory = () => DateTime.Now;
}...
```

```

`options.DefaultValueFactory` has a higher priority than the `options.DefaultValue` .

> Tip: Use `DefaultValueFactory` option only if the default value may change over the time (like `DateTime.Now` in this example). If it is a constant value, then use the `DefaultValue` option.

#### ## Validation

Entity extension system allows you to define validation for extension properties in a few ways.

#### #### Data Annotation Attributes

`Attributes` is a list of attributes associated to this property. The example code below adds two [data annotation validation attributes](<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation>) to the property:

```
```csharp
property =>
{
    property.Attributes.Add(new RequiredAttribute());
    property.Attributes.Add(new StringLengthAttribute(64) {MinimumLength =
4});
}
```

```

When you run the application, you see that the validation works out of the box:

![add-new-property-to-user-form](images/add-new-property-to-user-form-validation-error.png)

Since we've added the `RequiredAttribute`, it doesn't allow to left it blank. The validation system works;

- \* On the user interface (with automatic localization).
- \* On the HTTP API. Even if you directly perform an HTTP request, you get validation errors with a proper HTTP status code.
- \* On the ` SetProperty(...)` method on the entity (see [[the document](#)](Entities.md) if you wonder what is the ` SetProperty()` method).

So, it automatically makes a full stack validation.

> See the [ASP.NET Core MVC Validation document](<https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation>) to learn more about the attribute based validation.

#### ##### Default Validation Attributes

There are some attributes \*\*automatically added\*\* when you create certain type of properties;

- \* `RequiredAttribute` is added for \*\*non nullable\*\* primitive property types (e.g. `int`, `bool`, `DateTime`...) and `enum` types. If you want to allow nulls, make the property nullable (e.g. `int?`).
- \* `EnumDataTypeAttribute` is added for \*\*enum types\*\*, to prevent to set invalid enum values.

Use `property.Attributes.Clear();` if you don't want these attributes.

#### ##### Validation Actions

Validation actions allows you to execute a custom code to perform the validation. The example below checks if the `SocialSecurityNumber` starts with `B` and adds a validation error if so:

```
```csharp
property =>
{
    property.Attributes.Add(new RequiredAttribute());

```

```

        property.Attributes.Add(new StringLengthAttribute(64) {MinimumLength =
4});

        property.Validators.Add(context =>
{
    if (((string) context.Value).StartsWith("B"))
    {
        context.ValidationErrors.Add(
            new ValidationResult(
                "Social security number can not start with the letter
'B', sorry!",
                new[] {"extraProperties.SocialSecurityNumber"})
        );
    }
});
};

}...

```

Using a `'RegularExpressionAttribute'` might be better in this case, but this is just an example. Anyway, if you enter a value starts with the letter `'B'` you get the following error ****while saving the form****:

`![add-new-propert-to-user-form](images/add-new-property-to-user-form-validation-error-custom.png)`

The Context Object

The `'context'` object has useful properties that can be used in your custom validation action. For example, you can use the `'context.ServiceProvider'` to resolve services from the [\[dependency injection system\]](#)(Dependency-Injection.md). The example below gets the localizer and adds a localized error message:

```

```csharp
if (((string) context.Value).StartsWith("B"))
{
 var localizer = context.ServiceProvider
 .GetRequiredService<IStringLocalizer<MyProjectNameResource>>();

 context.ValidationErrors.Add(
 new ValidationResult(
 localizer["SocialSecurityNumberCannotStartWithB"],
 new[] {"extraProperties.SocialSecurityNumber"})
);
}
```

```

>`'context.ServiceProvider'` is nullable! It can be `'null'` only if you use the `'SetProperty(...)'` method on the object. Because DI system is not available on this time. While this is a rare case, you should perform a fallback logic when `'context.ServiceProvider'` is `'null'`. For this example, you would add a non-localized error message. This is not a problem since setting an invalid value to a property generally is a programmer mistake and you mostly don't need to localization in this case. In any way, you would not be able to use localization even in a regular property setter. But, if you are serious about

localization, you can throw a business exception (see the [exception handling document](<https://docs.abp.io/en/abp/latest/Exception-Handling>) to learn how to localize a business exception).

UI Visibility

When you define a property, it appears on the data table, create and edit forms on the related UI page. However, you can control each one individually. Example:

```
```csharp
property =>
{
 property.UI.OnTable.IsVisible = false;
 //...other configurations
}
````
```

Use `property.UI.OnCreateForm` and `property.UI.OnEditForm` to control forms too. If a property is required, but not added to the create form, you definitely get a validation exception, so use this option carefully. But a required property may not be in the edit form if that's your requirement.

UI Order

When you define a property, it appears on the data table, create and edit forms on the related UI page. However, you can control its order. Example:

```
```csharp
property =>
{
 property.UI.Order = 1;
 //...other configurations
}
````
```

Use `property.UI.OnCreateForm` and `property.UI.OnEditForm` to control forms too. If a property is required, but not added to the create form, you definitely get a validation exception, so use this option carefully. But a required property may not be in the edit form if that's your requirement.

HTTP API Availability

Even if you disable a property on UI, it can be still available through the HTTP API. By default, a property is available on all APIs.

Use the `property.Api` options to make a property unavailable in some API endpoints.

```
```csharp
property =>
{
 property.Api.OnUpdate.IsAvailable = false;
}
````
```

In this example, Update HTTP API will not allow to set a new value to this property. In this case, you also want to disable this property on the edit form:

```
```csharp
property =>
{
 property.Api.OnUpdate.IsAvailable = false;
 property.UI.OnEditForm.Visible = false;
}
...```

```

In addition to the `property.Api.OnUpdate`, you can set `property.Api.OnCreate` and `property.Api.OnGet` for a fine control the API endpoint.

## ## Special Types

### ### Enum

Module extension system naturally supports the `enum` types.

An example enum type:

```
```csharp
public enum UserType
{
    Regular,
    Moderator,
    SuperUser
}
...```

```

You can add enum properties just like others:

```
```csharp
user.AddOrUpdateProperty<UserType>("Type");
```

```

An enum properties is shown as combobox (select) in the create/edit forms:

![add-new-property-enum](images/add-new-property-enum.png)

Localization

Enum member name is shown on the table and forms by default. If you want to localize it, just create a new entry on your [localization](<https://docs.abp.io/en/abp/latest/Localization>) file:

```
```json
"Enum:UserType.0": "Super user"
```

```

One of the following names can be used as the localization key:

- * `Enum:UserType.0`
- * `Enum:UserType.SuperUser`
- * `UserType.0`

```
* `UserType.SuperUser`  
* `SuperUser`
```

Localization system searches for the key with the given order. Localized text are used on the table and the create/edit forms.

Navigation Properties / Foreign Keys

It is supported to add an extension property to an entity that is Id of another entity (foreign key).

Example: Associate a department to a user

```
```csharp  
ObjectExtensionManager.Instance.Modules()
 .ConfigureIdentity(identity =>
 {
 identity.ConfigureUser(user =>
 {
 user.AddOrUpdateProperty<Guid>(
 "DepartmentId",
 property =>
 {
 property.UI.Lookup.Url = "/api/departments";
 property.UI.Lookup.DisplayPropertyName = "name";
 }
);
 });
 ...);
...;
```

'UI.Lookup.Url' option takes a URL to get list of departments to select on edit/create forms. This endpoint can be a typical controller, an [auto API controller](API/Auto-API-Controllers.md) or any type of endpoint that returns a proper JSON response.

An example implementation that returns a fixed list of departments (in real life, you get the list from a data source):

```
```csharp  
[Route("api/departments")]  
public class DepartmentController : AbpController  
{  
    [HttpGet]  
    public async Task<ListResultDto<DepartmentDto>> GetAsync()  
    {  
        return new ListResultDto<DepartmentDto>(  
            new[]  
            {  
                new DepartmentDto  
                {  
                    Id = Guid.Parse("6267f0df-870f-4173-be44-d74b4b56d2bd"),  
                    Name = "Human Resources"  
                },  
                new DepartmentDto  
                {  
                    Id = Guid.Parse("21c7b61f-330c-489e-8b8c-80e0a78a5cc5"),  
                    Name = "Production"  
                }  
            }  
        );  
    }  
}
```

```

        }
    );
}
...

```

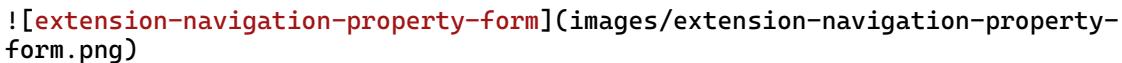
This API returns such a JSON response:

```

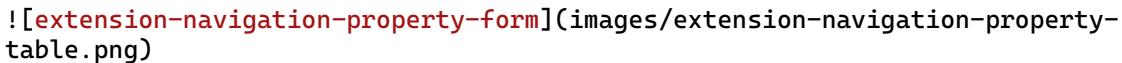
```json
{
 "items": [
 {
 "id": "6267f0df-870f-4173-be44-d74b4b56d2bd",
 "name": "Human Resources"
 },
 {
 "id": "21c7b61f-330c-489e-8b8c-80e0a78a5cc5",
 "name": "Production"
 }
]
}
...

```

ABP can now show an auto-complete select component to pick the department while creating or editing a user:

([extension-navigation-property-form](images/extension-navigation-property-form.png))

And shows the department name on the data table:

([extension-navigation-property-form](images/extension-navigation-property-table.png))

#### #### Lookup Options

`'UI.Lookup'` has the following options to customize how to read the response returned from the `'Url'`:

- \* `'Url'`: The endpoint to get the list of target entities. This is used on edit and create forms.
- \* `'DisplayPropertyName'`: The property in the JSON response to read the display name of the target entity to show on the UI. Default: `'text'`.
- \* `'ValuePropertyName'`: The property in the JSON response to read the Id of the target entity. Default: `'id'`.
- \* `'FilterParamName'`: ABP allows to search/filter the entity list on edit/create forms. This is especially useful if the target list contains a lot of items. In this case, you can return a limited list (top 100, for example) and allow user to search on the list. ABP sends filter text to the server (as a simple query string) with the name of this option. Default: `'filter'`.
- \* `'ResultListPropertyName'`: By default, returned JSON result should contain the entity list in an `'items'` array. You can change the name of this field. Default: `'items'`.

#### #### Lookup Properties: How Display Name Works?

You may wonder how ABP shows the department name on the data table above.

It is easy to understand how to fill the dropdown on edit and create forms: ABP makes an AJAX request to the given URL. It re-requests whenever user types to filter the items.

However, for the data table, multiple items are shown on the UI and performing a separate AJAX call to get display name of the department for each row would not be so efficient.

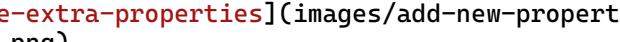
Instead, the display name of the foreign entity is also saved as an extra property of the entity (see *\*Extra Properties\** section of the [\[Entities\]\(Entities.md\)](#) document) in addition to Id of the foreign entity. If you check the database, you can see the '`DepartmentId_Text`' in the '`ExtraProperties`' field in the database table:

```
```json
{"DepartmentId": "21c7b61f-330c-489e-8b8c-
80e0a78a5cc5", "DepartmentId_Text": "Production"}
````
```

So, this is a type of *\*data duplication\**. If your target entity's name changes in the database later, there is no automatic synchronization system. The system works as expected, but you see the old name on the data tables. If that's a problem for you, you should care yourself to update this information when display name of your entity changes.

## ## Database Mapping

For relational databases, all extension property values are stored in a single field in the table:

![add-new-property-to-user-database-extra-properties](images/add-new-property-to-user-database-extra-properties.png)

'`ExtraProperties`' field stores the properties as a JSON object. While that's fine for some scenarios, you may want to create a dedicated field for your new property. Fortunately, it is very easy to configure.

If you are using the Entity Framework Core database provider, you can configure the database mapping as shown below:

```
```csharp
ObjectExtensionManager.Instance
    .MapEfCoreProperty<IdentityUser, string>(
        "SocialSecurityNumber",
        (entityBuilder, propertyBuilder) =>
    {
        propertyBuilder.HasMaxLength(64);
    }
);
````
```

Write this inside the '`YourProjectNameEfCoreEntityExtensionMappings`' class in your '`.EntityFrameworkCore`' project. Then you need to use the standard '`Add-Migration`' and '`Update-Database`' commands to create a new database migration and apply the change to your database.

Add-Migration create a new migration as shown below:

```

```csharp
public partial class Added_SocialSecurityNumber_To_IdentityUser : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.AddColumn<string>(
            name: "SocialSecurityNumber",
            table: "AbpUsers",
            maxLength: 128,
            nullable: true);
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropColumn(
            name: "SocialSecurityNumber",
            table: "AbpUsers");
    }
}
```

```

Once you update your database, you will see that the `AbpUsers` table has the new property as a standard table field:

[!\[add-new-propert-to-user-database-extra-properties\]\(images/add-new-propert-to-user-database-field.png\)](#)

> If you first created a property without a database table field, then you later needed to move this property to a database table field, it is suggested to execute an SQL command in your migration to copy the old values to the new field.  
 >  
 > However, if you don't make it, the ABP Framework seamlessly manages it. It uses the new database field, but fallbacks to the `ExtraProperties` field if it is null. When you save the entity, it moves the value to the new field.

See the [\[Extending Entities\]\(Customizing-Application-Modules-Extending-Entities.md\)](#) document for more.

[## More](#)

See the [\[Customizing the Modules\]\(Customizing-Application-Modules-Guide.md\)](#) guide for an overall index for all the extensibility options.

Here, a few things you can do:

- \* You can create a second entity that maps to the same database table with the extra property as a standard class property (if you've defined the EF Core mapping). For the example above, you can add a `public string SocialSecurityNumber {get; set;}` property to the `AppUser` entity in your application, since the `AppUser` entity is mapped to the same `AbpUser` table. Do this only if you need it, since it brings more complexity to your application.
- \* You can override a domain or application service to perform custom logics with your new property.
- \* You can low level control how to add/render a field in the data table on the UI.

## ## See Also

- \* [Angular UI Extensions](UI/Angular/Extensions-Overall.md)

### 8.1.4.3 Customizing/Extending Entities

# Customizing the Application Modules: Extending Entities

In some cases, you may want to add some additional properties (and database fields) for an entity defined in a depended module. This section will cover some different approaches to make this possible.

#### ## Extra Properties

[Extra properties](Entities.md) is a way of storing some additional data on an entity without changing it. The entity should implement the `IHasExtraProperties` interface to allow it. All the aggregate root entities defined in the pre-built modules implement the `IHasExtraProperties` interface, so you can store extra properties on these objects.

Example:

```
```csharp
//SET AN EXTRA PROPERTY
var user = await _identityUserRepository.GetAsync(userId);
user SetProperty("Title", "My custom title value!");
await _identityUserRepository.UpdateAsync(user);

//GET AN EXTRA PROPERTY
var user = await _identityUserRepository.GetAsync(userId);
return user.GetProperty<string>("Title");
```
```

This approach is very easy to use and available out of the box. No extra code needed. You can store more than one property at the same time by using different property names (like `Title` here).

Extra properties are stored as a single `JSON` formatted string value in the database for the EF Core. For MongoDB, they are stored as separate fields of the document.

See the [entities document](Entities.md) for more about the extra properties system.

> It is possible to perform a \*\*business logic\*\* based on the value of an extra property. You can [override a service method](Customizing-Application-Modules-Overriding-Services.md), then get or set the value as shown above.

#### ## Entity Extensions (EF Core)

As mentioned above, all extra properties of an entity are stored as a single JSON object in the database table. This is not so natural especially when you want to;

\* Create \*\*indexes\*\* and \*\*foreign keys\*\* for an extra property.

- \* Write \*\*SQL\*\* or \*\*LINQ\*\* using the extra property (search table by the property value, for example).
- \* Creating your \*\*own entity\*\* maps to the same table, but defines an extra property as a \*\*regular property\*\* in the entity (see the [EF Core migration document](Entity-Framework-Core-Migrations.md) for more).

To overcome the difficulties described above, ABP Framework entity extension system for the Entity Framework Core that allows you to use the same extra properties API defined above, but store a desired property as a separate field in the database table.

Assume that you want to add a `SocialSecurityNumber` to the `IdentityUser` entity of the [Identity Module](Modules/Identity.md). You can use the `ObjectExtensionManager`:

```
```csharp
ObjectExtensionManager.Instance
    .MapEfCoreProperty<IdentityUser, string>(
        "SocialSecurityNumber",
        (entityBuilder, propertyBuilder) =>
    {
        propertyBuilder.HasMaxLength(32);
    }
);
...;
```

- * You provide the `IdentityUser` as the entity name, `string` as the type of the new property, `SocialSecurityNumber` as the property name (also, the field name in the database table).
- * You also need to provide an action that defines the database mapping properties using the [EF Core Fluent API](https://docs.microsoft.com/en-us/ef/core/modeling/entity-properties).

> This code part must be executed before the related `DbContext` used. The [application startup template](Startup-Templates/Application.md) defines a static class named `YourProjectNameEfCoreEntityExtensionMappings`. You can define your extensions in this class to ensure that it is executed in the proper time. Otherwise, you should handle it yourself.

Once you define an entity extension, you then need to use the standard [Add-Migration](https://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/powershell#add-migration) and [Update-Database](https://docs.microsoft.com/en-us/ef/core/miscellaneous/cli/powershell#update-database) commands of the EF Core to create a code first migration class and update your database.

You can then use the same extra properties system defined in the previous section to manipulate the property over the entity.

Creating a New Entity Maps to the Same Database Table/Collection

Another approach can be **creating your own entity** mapped to **the same database table** (or collection for a MongoDB database).

Creating a New Entity with Its Own Database Table/Collection

Mapping your entity to an **existing table** of a depended module has a few disadvantages;

- * You deal with the **database migration structure** for EF Core. While it is possible, you should extra care about the migration code especially when you want to add **relations** between entities.
- * Your application database and the module database will be the **same physical database**. Normally, a module database can be separated if needed, but using the same table restricts it.

If you want to **loose couple** your entity with the entity defined by the module, you can create your own database table/collection and map your entity to your own table in your own database.

In this case, you need to deal with the **synchronization problems**, especially if you want to **duplicate** some properties/fields of the related entity. There are a few solutions;

- * If you are building a **monolithic** application (or managing your entity and the related module entity within the same process), you can use the [local event bus](Local-Event-Bus.md) to listen changes.
- * If you are building a **distributed** system where the module entity is managed (created/updated/deleted) on a different process/service than your entity is managed, then you can subscribe to the [distributed event bus](Distributed-Event-Bus.md) for change events.

Once you handle the event, you can update your own entity in your own database.

[### Subscribing to Local Events](#)

[Local Event Bus](Local-Event-Bus.md) system is a way to publish and subscribe to events occurring in the same application.

Assume that you want to get informed when a '`IdentityUser`' entity changes (created, updated or deleted). You can create a class that implements the '`ILocalEventHandler<EntityChangedEventArgs<IdentityUser>>`' interface.

```
```csharp
public class MyLocalIdentityUserChangeEventHandler :
 ILocalEventHandler<EntityChangedEventArgs<IdentityUser>>,
 ITransientDependency
{
 public async Task HandleEventAsync(EntityChangedEventArgs<IdentityUser>
eventData)
 {
 var userId = eventData.Entity.Id;
 var userName = eventData.Entity.UserName;

 //...
 }
}
```

```

- * '`EntityChangedEventArgs<T>`' covers create, update and delete events for the given entity. If you need, you can subscribe to create, update and delete events individually (in the same class or different classes).
- * This code will be executed in the **current unit of work**, the whole process becomes transactional.

> Reminder: This approach needs to change the `IdentityUser` entity in the same process contains the handler class. It perfectly works even for a clustered environment (when multiple instances of the same application are running on multiple servers).

Subscribing to Distributed Events

[Distributed Event Bus](Distributed-Event-Bus.md) system is a way to publish an event in one application and receive the event in the same or different application running on the same or different server.

Assume that you want to get informed when `Tenant` entity (of the [Tenant Management](Modules/Tenant-Management.md) module) has created. In this case, you can subscribe to the `EntityCreatedEto<TenantEto>` event as shown in the following example:

```
```csharp
public class MyDistributedEventHandler :
 IDistributedEventHandler<EntityCreatedEto<TenantEto>>,
 ITransientDependency
{
 public async Task HandleEventAsync(EntityCreatedEto<TenantEto> eventData)
 {
 var tenantId = eventData.Entity.Id;
 var tenantName = eventData.Entity.Name;
 //...your custom logic
 }

 //...
}
```

```

This handler is executed only when a new tenant has been created. All the pre-built ABP [application modules](Modules/Index.md) define corresponding `ETO` types for their entities. So, you can easily get informed when they changes.

> Notice that ABP doesn't publish distributed events for an entity by default. Because it has a cost and should be enabled by intention. See the [distributed event bus document](Distributed-Event-Bus.md) to learn more.

See Also

- * [Migration System for the EF Core](Entity-Framework-Core-Migrations.md)
- * [Customizing the Existing Modules](Customizing-Application-Modules-Guide.md)

8.1.4.4 Customizing/Overriding Services

Customizing the Application Modules: Overriding Services

You may need to **change behavior (business logic)** of a depended module for your application. In this case, you can use the power of the [dependency injection system](Dependency-Injection.md) to replace a service, controller or even a page model of the depended module by your own implementation.

Replacing a service is possible for any type of class registered to the dependency injection, including services of the ABP Framework.

You have different options can be used based on your requirement those will be explained in the next sections.

> Notice that some service methods may not be virtual, so you may not be able to override. We make all virtual by design. If you find any method that is not overridable, please [create an issue](<https://github.com/abpframework/abp/issues/new>) or do it yourself and send a **pull request** on GitHub.

Replacing an Interface

If given service defines an interface, like the `IdentityUserAppService` class implements the `IIdentityUserAppService`, you can re-implement the same interface and replace the current implementation by your class. Example:

```
```csharp
public class MyIdentityUserAppService : IIdentityUserAppService,
ITransientDependency
{
 //...
}
````
```

`MyIdentityUserAppService` replaces the `IIdentityUserAppService` by naming convention (since both ends with `IdentityUserAppService`). If your class name doesn't match, you need to manually expose the service interface:

```
```csharp
[ExposeServices(typeof(IIdentityUserAppService))]
public class TestAppService : IIdentityUserAppService, ITransientDependency
{
 //...
}
````
```

The dependency injection system allows to register multiple services for the same interface. The last registered one is used when the interface is injected. It is a good practice to explicitly replace the service.

Example:

```
```csharp
[Dependency(ReplaceServices = true)]
[ExposeServices(typeof(IIdentityUserAppService))]
public class TestAppService : IIdentityUserAppService, ITransientDependency
{
 //...
}
````
```

In this way, there will be a single implementation of the `IIdentityUserAppService` interface, while it doesn't change the result for this case. Replacing a service is also possible by code:

```
```csharp
```

```
context.Services.Replace(
 ServiceDescriptor.Transient<IIdentityUserAppService,
 MyIdentityUserAppService>()
);...
```

You can write this inside the `ConfigureServices` method of your [module](Module-Development-Basics.md).

## ## Overriding a Service Class

In most cases, you will want to change one or a few methods of the current implementation for a service. Re-implementing the complete interface would not be efficient in this case. As a better approach, inherit from the original class and override the desired method.

### ### Example: Overriding an Application Service

```
```csharp
[Dependency(ReplaceServices = true)]
[ExposeServices(typeof(IIdentityUserAppService),
    typeof(IdentityUserAppService), typeof(MyIdentityUserAppService))]
public class MyIdentityUserAppService : IdentityUserAppService
{
    //...
    public MyIdentityUserAppService(
        IdentityUserManager userManager,
        IIdentityUserRepository userRepository,
        IGuidGenerator guidGenerator
    ) : base(
        userManager,
        userRepository,
        guidGenerator)
    {
    }

    public async override Task<IdentityUserDto>
CreateAsync(IdentityUserCreateDto input)
    {
        if (input.PhoneNumber.IsNullOrWhiteSpace())
        {
            throw new AbpValidationException(
                "Phone number is required for new users!",
                new List<ValidationResult>
                {
                    new ValidationResult(
                        "Phone number can not be empty!",
                        new []{"PhoneNumber"}
                    )
                }
            );
        }

        return await base.CreateAsync(input);
    }
}...```

```

This class **overrides** the `CreateAsync` method of the `IdentityUserAppService` [application service](Application-Services.md) to check the phone number. Then calls the base method to continue to the **underlying business logic**. In this way, you can perform additional business logic **before** and **after** the base logic.

You could completely **re-write** the entire business logic for a user creation without calling the base method.

Example: Overriding a Domain Service

```
```csharp
[Dependency(ReplaceServices = true)]
[ExposeServices(typeof(IdentityUserManager))]
public class MyIdentityUserManager : IdentityUserManager
{
 public MyIdentityUserManager(
 IdentityUserStore store,
 IIdentityRoleRepository roleRepository,
 IIdentityUserRepository userRepository,
 IOptions<IdentityOptions> optionsAccessor,
 IPasswordHasher<IdentityUser> passwordHasher,
 IEnumerable<IUserValidator<IdentityUser>> userValidators,
 IEnumerable<IPasswordValidator<IdentityUser>> passwordValidators,
 ILookupNormalizer keyNormalizer,
 IdentityErrorDescriber errors,
 IServiceProvider services,
 ILogger<IdentityUserManager> logger,
 ICancellationTokenProvider cancellationTokenProvider) :
 base(store,
 roleRepository,
 userRepository,
 optionsAccessor,
 passwordHasher,
 userValidators,
 passwordValidators,
 keyNormalizer,
 errors,
 services,
 logger,
 cancellationTokenProvider)
 {
 }

 public async override Task<IdentityResult> CreateAsync(IdentityUser user)
 {
 if (user.PhoneNumber.IsNullOrWhiteSpace())
 {
 throw new AbpValidationException(
 "Phone number is required for new users!",
 new List<ValidationResult>
 {
 new ValidationResult(
 "Phone number can not be empty!",
 new []{"PhoneNumber"})
 }
);
 }
 }
}
```

```

 }

 return await base.CreateAsync(user);
 }
}
```

```

This example class inherits from the `IdentityUserManager` [domain service](Domain-Services.md) and overrides the `CreateAsync` method to perform the same phone number check implemented above. The result is same, but this time we've implemented it inside the domain service assuming that this is a **core domain logic** for our system.

> `[ExposeServices(typeof(IdentityUserManager))]` attribute is **required** here since `IdentityUserManager` does not define an interface (like `IIdentityUserManager`) and dependency injection system doesn't expose services for inherited classes (like it does for the implemented interfaces) by convention.

Check the [localization system](Localization.md) to learn how to localize the error messages.

Example: Overriding a Repository

```

```csharp
public class MyEfCoreIdentityUserRepository : EfCoreIdentityUserRepository
{
 public MyEfCoreIdentityUserRepository(
 IDbContextProvider<IIdentityDbContext> dbContextProvider)
 : base(dbContextProvider)
 {
 }

 /* You can override any base method here */
}
```

```

In this example, we are overriding the `EfCoreIdentityUserRepository` class that is defined by the [Identity module](Modules/Identity.md). This is the [Entity Framework Core](Entity-Framework-Core.md) implementation of the user repository.

Thanks to the naming convention (`MyEfCoreIdentityUserRepository` ends with `EfCoreIdentityUserRepository`), no additional setup is required. You can override any base method to customize it for your needs.

However, if you inject ` IRepository<IdentityUser>` or ` IRepository<IdentityUser, Guid>`, it will still use the default repository implementation. To replace the default repository implementation, write the following code in the `ConfigureServices` method of your module class:

```

```csharp
context.Services.AddDefaultRepository(
 typeof(Volo.Abp.Identity.IdentityUser),
 typeof(MyEfCoreIdentityUserRepository),
 replaceExisting: true
);
```

```

In this way, your implementation will be used if you inject ` IRepository<IdentityUser>` , ` IRepository<IdentityUser, Guid>` or ` IIdentityUserRepository` .

If you want to add extra methods to your repository and use it in your own code, you can define an interface and expose it from your repository implementation. You can also extend the pre-built repository interface. Example:

```
```csharp
public interface IMyIdentityUserRepository : IIdentityUserRepository
{
 public Task DeleteByEmail(string email);
}
...```

```

The `IMyIdentityUserRepository` interface extends the Identity module's `IIdentityUserRepository` interface. Then you can implement it as shown in the following example:

```
```csharp
[ExposeServices(typeof(IMyIdentityUserRepository), IncludeDefaults = true)]
public class MyEfCoreIdentityUserRepository
    : EfCoreIdentityUserRepository, IMyIdentityUserRepository
{
    public MyEfCoreIdentityUserRepository(
        IDbContextProvider<IIdentityDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }

    public async Task DeleteByEmail(string email)
    {
        var dbContext = await GetDbContextAsync();
        var user = await dbContext.Users.FirstOrDefaultAsync(u => u.Email == email);
        if (user != null)
        {
            dbContext.Users.Remove(user);
        }
    }
}
...```

```

The `MyEfCoreIdentityUserRepository` class implements the `IMyIdentityUserRepository` interface. `ExposeServices` attribute is needed since ABP can not expose `IMyIdentityUserRepository` by naming conventions (`MyEfCoreIdentityUserRepository` doesn't end with `MyIdentityUserRepository`). Now, you can inject the `IMyIdentityUserRepository` interface into your services and call its `DeleteByEmail` method.

Example: Overriding a Controller

```
```csharp
using System.Threading.Tasks;
using Microsoft.Extensions.Logging;
```

```

using Volo.Abp.Account;
using Volo.Abp.DependencyInjection;

namespace MyProject.Controllers
{
 [Dependency(ReplaceServices = true)]
 [ExposeServices(typeof(AccountController))]
 public class MyAccountController : AccountController
 {
 public MyAccountController(IAccountAppService accountAppService)
 : base(accountAppService)
 {

 }

 public async override Task SendPasswordResetCodeAsync(
 SendPasswordResetCodeDto input)
 {
 Logger.LogInformation("Your custom logic...");
 await base.SendPasswordResetCodeAsync(input);
 }
 }
}
```

```

This example replaces the `AccountController` (An API Controller defined in the [\[Account Module\]](#)(Modules/Account.md)) and overrides the `SendPasswordResetCodeAsync` method.

****`[ExposeServices(typeof(AccountController))]*` is essential** here since it registers this controller for the `AccountController` in the dependency injection system. `[Dependency(ReplaceServices = true)]` is also recommended to clear the old registration (even the ASP.NET Core DI system selects the last registered one).**

In addition, the `MyAccountController` will be removed from [\[`ApplicationModel`\]\(https://docs.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.mvc.applicationmodels.applicationmodel.controllers\)](#) because it defines `ExposeServicesAttribute`.

If `IncludeSelf = true` is specified, i.e. `[ExposeServices(typeof(AccountController), IncludeSelf = true)]`, then `AccountController` will be removed instead. This is useful for **extending** a controller.

If you don't want to remove either controller, you can configure `AbpAspNetCoreMvcOptions`:

```

```csharp
Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.IgnoredControllersOnModelExclusion
 .AddIfNotContains(typeof(MyAccountController));
});
```

```

Overriding Other Classes

Overriding controllers, framework services, view component classes and any other type of classes registered to dependency injection can be overridden just like the examples above.

Extending Data Transfer Objects

Extending [entities](Entities.md) is possible as described in the [Extending Entities document](Customizing-Application-Modules-Extending-Entities.md). In this way, you can add **custom properties** to entities and perform **additional business logic** by overriding the related services as described above.

It is also possible to extend Data Transfer Objects (**DTOs**) used by the application services. In this way, you can get extra properties from the UI (or client) and return extra properties from the service.

Example

Assuming that you've already added a 'SocialSecurityNumber' as described in the [Extending Entities document](Customizing-Application-Modules-Extending-Entities.md) and want to include this information while getting the list of users from the 'GetListAsync' method of the 'IdentityUserAppService'.

You can use the [object extension system](Object-Extensions.md) to add the property to the 'IdentityUserDto'. Write this code inside the 'YourProjectNameDtoExtensions' class comes with the application startup template:

```
```csharp
ObjectExtensionManager.Instance
 .AddOrUpdateProperty<IdentityUserDto, string>(
 "SocialSecurityNumber"
);
...```

```

This code defines a 'SocialSecurityNumber' to the 'IdentityUserDto' class as a 'string' type. That's all. Now, if you call the '/api/identity/users' HTTP API (which uses the 'IdentityUserAppService' internally) from a REST API client, you will see the 'SocialSecurityNumber' value in the 'extraProperties' section.

```
```json
{
    "totalCount": 1,
    "items": [
        {
            "tenantId": null,
            "userName": "admin",
            "name": "admin",
            "surname": null,
            "email": "admin@abp.io",
            "emailConfirmed": false,
            "phoneNumber": null,
            "phoneNumberConfirmed": false,
            "twoFactorEnabled": false,
            "lockoutEnabled": true,
            "lockoutEnd": null,
            "concurrencyStamp": "b4c371a0ab604de28af472fa79c3b70c",
        }
    ]
}```
```

```

        "isDeleted": false,
        "deleterId": null,
        "deletionTime": null,
        "lastModificationTime": "2020-04-09T21:25:47.0740706",
        "lastModifierId": null,
        "creationTime": "2020-04-09T21:25:46.8308744",
        "creatorId": null,
        "id": "8edecb8f-1894-a9b1-833b-39f4725db2a3",
        "extraProperties": {
            "SocialSecurityNumber": "123456789"
        }
    }
}
```

```

Manually added the `123456789` value to the database for now.

All pre-built modules support extra properties in their DTOs, so you can configure easily.

### ### Definition Check

When you [define](Customizing-Application-Modules-Extending-Entities.md) an extra property for an entity, it doesn't automatically appear in all the related DTOs, because of the security. The extra property may contain a sensitive data and you may not want to expose it to the clients by default.

So, you need to explicitly define the same property for the corresponding DTO if you want to make it available for the DTO (as just done above). If you want to allow to set it on user creation, you also need to define it for the `IdentityUserCreateDto`.

If the property is not so secure, this can be tedious. Object extension system allows you to ignore this definition check for a desired property. See the example below:

```

```csharp
ObjectExtensionManager.Instance
    .AddOrUpdateProperty<IdentityUser, string>(
        "SocialSecurityNumber",
        options =>
    {
        options.MapEfCore(b => bHasMaxLength(32));
        options.CheckPairDefinitionOnMapping = false;
    }
);
```

```

This is another approach to define a property for an entity (`ObjectExtensionManager` has more, see [its document](Object-Extensions.md)). This time, we set `CheckPairDefinitionOnMapping` to false to skip definition check while mapping entities to DTOs and vice versa.

If you don't like this approach but want to add a single property to multiple objects (DTOs) easier, `AddOrUpdateProperty` can get an array of types to add the extra property:

```

```csharp

```

```

ObjectExtensionManager.Instance
    .AddOrUpdateProperty<string>(
        new[]
        {
            typeof(IdentityUserDto),
            typeof(IdentityUserCreateDto),
            typeof(IdentityUserUpdateDto)
        },
        "SocialSecurityNumber"
    );
...

```

About the User Interface

This system allows you to add extra properties to entities and DTOs and execute custom business code, however it does nothing related to the User Interface.

See [[Overriding the User Interface](#)](Customizing-Application-Modules-Overriding-User-Interface.md) guide for the UI part.

How to Find the Services?

[[Module documents](#)](Modules/Index.md) includes the list of the major services they define. In addition, you can investigate [[their source code](#)](<https://github.com/abpframework/abp/tree/dev/modules>) to explore all the services.

8.2 Domain Driven Design

8.2.1 Overall

Domain Driven Design

What is DDD?

ABP framework provides an **infrastructure** to make **Domain Driven Design** based development easier to implement. DDD is [[defined in the Wikipedia](#)](https://en.wikipedia.org/wiki/Domain-driven_design) as below:

- > **Domain-driven design** (**DDD**) is an approach to software development for complex needs by connecting the implementation to an evolving model. The premise of domain-driven design is the following:
 - >
 - > - Placing the project's primary focus on the core domain and domain logic;
 - > - Basing complex designs on a model of the domain;
 - > - Initiating a creative collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems.

Layers & Building Blocks

ABP follows DDD principles and patterns to achieve a layered application model which consists of four fundamental layers:

- **Presentation Layer**: Provides an interface to the user. Uses the **Application Layer** to achieve user interactions.
- **Application Layer**: Mediates between the Presentation and Domain Layers. Orchestrates business objects to perform specific application tasks. Implements use cases as the application logic.
- **Domain Layer**: Includes business objects and the core (domain) business rules. This is the heart of the application.
- **Infrastructure Layer**: Provides generic technical capabilities that support higher layers mostly using 3rd-party libraries.

DDD mostly interest in the **Domain** and the **Application** layers, rather than the Infrastructure and the Presentation layers. The following documents explains the **infrastructure** provided by the ABP Framework to implement **Building Blocks** of the DDD:

- * **Domain Layer**
 - * [Entities & Aggregate Roots](Entities.md)
 - * [Repositories](Repositories.md)
 - * [Domain Services](Domain-Services.md)
 - * [Value Objects](Value-Objects.md)
 - * [Specifications](Specifications.md)
- * **Application Layer**
 - * [Application Services](Application-Services.md)
 - * [Data Transfer Objects (DTOs)](Data-Transfer-Objects.md)
 - * [Unit of Work](Unit-Of-Work.md)

Free E-Book: Implementing DDD

See the [\[Implementing Domain Driven Design book\]](https://abp.io/books/implementing-domain-driven-design)(<https://abp.io/books/implementing-domain-driven-design>) as a **complete reference**. This book explains the Domain Driven Design and introduces explicit **rules and examples** to give a deep understanding of the **implementation details**.

8.2.2 Domain Layer

8.2.2.1 Entities & Aggregate Roots

Entities

Entities are one of the core concepts of DDD (Domain Driven Design). Eric Evans describes it as "*An object that is not fundamentally defined by its attributes, but rather by a thread of continuity and identity*".

An entity is generally mapped to a table in a relational database.

Entity Class

Entities are derived from the `Entity<TKey>` class as shown below:

```
```C#
public class Book : Entity<Guid>
{
 public string Name { get; set; }

 public float Price { get; set; }
}
```

...

> If you do not want to derive your entity from the base `Entity<TKey>` class, you can directly implement `IEntity<TKey>` interface.

`Entity<TKey>` class just defines an `Id` property with the given primary \*\*key type\*\*, which is `Guid` in the example above. It can be other types like `string`, `int`, `long`, or whatever you need.

### ### Entities with GUID Keys

If your entity's Id type is `Guid`, there are some good practices to implement:

- \* Create a constructor that gets the Id as a parameter and passes to the base class.
  - \* If you don't set a GUID Id, \*\*ABP Framework sets it on save\*\*, but it is good to have a valid Id on the entity even before saving it to the database.
  - \* If you create an entity with a constructor that takes parameters, also create a `private` or `protected` empty constructor. This is used while your database provider reads your entity from the database (on deserialization).
  - \* Don't use the `Guid.NewGuid()` to set the Id! \*\*Use [the `IGuidGenerator` service](Guid-Generation.md)\*\* while passing the Id from the code that creates the entity. `IGuidGenerator` optimized to generate sequential GUIDs, which is critical for clustered indexes in the relational databases.

An example entity:

```
```csharp
public class Book : Entity<Guid>
{
    public string Name { get; set; }

    public float Price { get; set; }

    protected Book()
    {

    }

    public Book(Guid id)
        : base(id)
    {

    }
}
````
```

Example usage in an [application service](Application-Services.md):

```
```csharp
public class BookAppService : ApplicationService, IBookAppService
{
    private readonly IRepository<Book> _bookRepository;

    public BookAppService(IRepository<Book> bookRepository)
    {
        _bookRepository = bookRepository;
    }
}
```

```

    }

    public async Task CreateAsync(CreateBookDto input)
    {
        await _bookRepository.InsertAsync(
            new Book(GuidGenerator.Create())
            {
                Name = input.Name,
                Price = input.Price
            }
        );
    }
}...

```

* `BookAppService` injects the default [repository](Repositories.md) for the book entity and uses its `InsertAsync` method to insert a `Book` to the database.

* `GuidGenerator` is type of `IGuidGenerator` which is a property defined in the `ApplicationService` base class. ABP defines such frequently used base properties as pre-injected for you, so you don't need to manually [inject](Dependency-Injection.md) them.

* If you want to follow the DDD best practices, see the **Aggregate Example** section below.

Entities with Composite Keys

Some entities may need to have **composite keys**. In that case, you can derive your entity from the non-generic `Entity` class. Example:

```
```C#
public class UserRole : Entity
{
 public Guid UserId { get; set; }

 public Guid RoleId { get; set; }

 public DateTime CreationTime { get; set; }

 public UserRole()
 {

 }

 public override object[] GetKeys()
 {
 return new object[] { UserId, RoleId };
 }
}...

```

For the example above, the composite key is composed of `UserId` and `RoleId`. For a relational database, it is the composite primary key of the related table. Entities with composite keys should implement the `GetKeys()` method as shown above.

> Notice that you also need to define keys of the entity in your \*\*object-relational mapping\*\* (ORM) configuration. See the [Entity Framework Core](Entity-Framework-Core.md) integration document for example.

> Also note that Entities with Composite Primary Keys cannot utilize the ` IRepository< TEntity, TKey >` interface since it requires a single Id property. However, you can always use ` IRepository< TEntity >`. See [repositories documentation](Repositories.md) for more.

### ### EntityEquals

` Entity.EntityEquals(...)` method is used to check if two Entity Objects are equals.

Example:

```
```csharp
Book book1 = ...
Book book2 = ...

if (book1.EntityEquals(book2)) //Check equality
{
    ...
}
```

```

### ## AggregateRoot Class

" \*Aggregate is a pattern in Domain-Driven Design. A DDD aggregate is a cluster of domain objects that can be treated as a single unit. An example may be an order and its line-items, these will be separate objects, but it's useful to treat the order (together with its line items) as a single aggregate.\*" (see the [full description](http://martinfowler.com/bliki/DDD\_Aggregate.html))

` AggregateRoot< TKey >` class extends the ` Entity< TKey >` class. So, it also has an ` Id ` property by default.

> Notice that ABP creates default repositories only for aggregate roots by default. However, it's possible to include all entities. See the [repositories documentation](Repositories.md) for more.

ABP does not force you to use aggregate roots, you can in fact use the ` Entity ` class as defined before. However, if you want to implement the [Domain Driven Design](Domain-Driven-Design.md) and want to create aggregate root classes, there are some best practices you may want to consider:

- \* An aggregate root is responsible to preserve it's own integrity. This is also true for all entities, but aggregate root has responsibility for it's sub entities too. So, the aggregate root must always be in a valid state.
- \* An aggregate root can be referenced by it's Id. Do not reference it by it's navigation property.
- \* An aggregate root is treated as a single unit. It's retrieved and updated as a single unit. It's generally considered as a transaction boundary.
- \* Work with sub-entities over the aggregate root- do not modify them independently.

See the [entity design best practice guide](Best-Practices/Entities.md) if you want to implement DDD in your application.

### ### Aggregate Example

This is a full sample of an aggregate root with a related sub-entity collection:

```
```C#
public class Order : AggregateRoot<Guid>
{
    public virtual string ReferenceNo { get; protected set; }

    public virtual int TotalItemCount { get; protected set; }

    public virtual DateTime CreationTime { get; protected set; }

    public virtual List<OrderLine> OrderLines { get; protected set; }

    protected Order()
    {

    }

    public Order(Guid id, string referenceNo)
    {
        Check.NotNull(referenceNo, nameof(referenceNo));

        Id = id;
        ReferenceNo = referenceNo;

        OrderLines = new List<OrderLine>();
    }

    public void AddProduct(Guid productId, int count)
    {
        if (count <= 0)
        {
            throw new ArgumentException(
                "You can not add zero or negative count of products!",
                nameof(count)
            );
        }

        var existingLine = OrderLines.FirstOrDefault(ol => ol.ProductId == productId);

        if (existingLine == null)
        {
            OrderLines.Add(new OrderLine(this.Id, productId, count));
        }
        else
        {
            existingLine.ChangeCount(existingLine.Count + count);
        }

        TotalItemCount += count;
    }
}
```

```

}

public class OrderLine : Entity
{
    public virtual Guid OrderId { get; protected set; }

    public virtual Guid ProductId { get; protected set; }

    public virtual int Count { get; protected set; }

    protected OrderLine()
    {

    }

    internal OrderLine(Guid orderId, Guid productId, int count)
    {
        OrderId = orderId;
        ProductId = productId;
        Count = count;
    }

    internal void ChangeCount(int newCount)
    {
        Count = newCount;
    }

    public override object[] GetKeys()
    {
        return new Object[] {OrderId, ProductId};
    }
}
...

```

> If you do not want to derive your aggregate root from the base `AggregateRoot< TKey >` class, you can directly implement the `IAggregateRoot< TKey >` interface.

`Order` is an **aggregate root** with `Guid` type `Id` property. It has a collection of `OrderLine` entities. `OrderLine` is another entity with a composite primary key (`OrderId` and `ProductId`).

While this example may not implement all the best practices of an aggregate root, it still follows some good practices:

- * `Order` has a public constructor that takes **minimal requirements** to construct an `Order` instance. So, it's not possible to create an order without an id and reference number. The **protected/private** constructor is only necessary to **deserialize** the object while reading from a data source.
- * `OrderLine` constructor is internal, so it is only allowed to be created by the domain layer. It's used inside of the `Order.AddProduct` method.
- * `Order.AddProduct` implements the business rule to add a product to an order.
- * All properties have `protected` setters. This is to prevent the entity from arbitrary changes from outside of the entity. For example, it would be dangerous to set `TotalItemCount` without adding a new product to the order. Its value is maintained by the `AddProduct` method.

ABP Framework does not force you to apply any DDD rule or patterns. However, it tries to make it possible and easier when you do want to apply them. The documentation also follows the same principle.

Aggregate Roots with Composite Keys

While it's not common (and not suggested) for aggregate roots, it is in fact possible to define composite keys in the same way as defined for the mentioned entities above. Use non-generic `'AggregateRoot'` base class in that case.

BasicAggregateRoot Class

`'AggregateRoot'` class implements the `'IHasExtraProperties'` and `'IHasConcurrencyStamp'` interfaces which brings two properties to the derived class. `'IHasExtraProperties'` makes the entity extensible (see the **Extra Properties** section below) and `'IHasConcurrencyStamp'` adds a `'ConcurrencyStamp'` property that is managed by the ABP Framework to implement the [optimistic concurrency](<https://docs.microsoft.com/en-us/ef/core/saving/concurrency>). In most cases, these are wanted features for aggregate roots.

However, if you don't need these features, you can inherit from the `'BasicAggregateRoot< TKey >'` (or `'BasicAggregateRoot'`) for your aggregate root.

Base Classes & Interfaces for Audit Properties

There are some properties like `'CreationTime'`, `'CreatorId'`, `'LastModificationTime'`... which are very common in all applications. ABP Framework provides some interfaces and base classes to **standardize** these properties and also **sets their values automatically**.

Auditing Interfaces

There are a lot of auditing interfaces, so you can implement the one that you need.

> While you can manually implement these interfaces, you can use **the base classes** defined in the next section to simplify it.

- * `'IHasCreationTime'` defines the following properties:
 - * `'CreationTime'`
- * `'IMayHaveCreator'` defines the following properties:
 - * `'CreatorId'`
- * `'ICreationAuditedObject'` inherits from the `'IHasCreationTime'` and the `'IMayHaveCreator'`, so it defines the following properties:
 - * `'CreationTime'`
 - * `'CreatorId'`
- * `'IHasModificationTime'` defines the following properties:
 - * `'LastModificationTime'`
- * `'IModificationAuditedObject'` extends the `'IHasModificationTime'` and adds the `'LastModifierId'` property. So, it defines the following properties:
 - * `'LastModificationTime'`
 - * `'LastModifierId'`
- * `'IAuditedObject'` extends the `'ICreationAuditedObject'` and the `'IModificationAuditedObject'`, so it defines the following properties:
 - * `'CreationTime'`

```

* `CreatorId`
* `LastModificationTime`
* `LastModifierId`
* `ISoftDelete` (see the [data filtering document](Data-Filtering.md))
defines the following properties:
* `IsDeleted`
* `IHasDeletionTime` extends the `ISoftDelete` and adds the `DeletionTime` property. So, it defines the following properties:
* `IsDeleted`
* `DeletionTime`
* `IDeleteAuditedObject` extends the `IHasDeletionTime` and adds the `DeleterId` property. So, it defines the following properties:
* `IsDeleted`
* `DeletionTime`
* `DeleterId`
* `IFullAuditedObject` inherits from the `IAuditedObject` and the `IDeleteAuditedObject`, so it defines the following properties:
* `CreationTime`
* `CreatorId`
* `LastModificationTime`
* `LastModifierId`
* `IsDeleted`
* `DeletionTime`
* `DeleterId`

```

Once you implement any of the interfaces, or derive from a class defined in the next section, ABP Framework automatically manages these properties wherever possible.

> Implementing `ISoftDelete`, `IDeleteAuditedObject` or `IFullAuditedObject` makes your entity **soft-delete**. See the [data filtering document](Data-Filtering.md) to learn about the soft-delete pattern.

[### Auditing Base Classes](#)

While you can manually implement any of the interfaces defined above, it is suggested to inherit from the base classes defined here:

- * `CreationAuditedEntity< TKey >` and `CreationAuditedAggregateRoot< TKey >` implement the `ICreationAuditedObject` interface.
- * `AuditedEntity< TKey >` and `AuditedAggregateRoot< TKey >` implement the `IAuditedObject` interface.
- * `FullAuditedEntity< TKey >` and `FullAuditedAggregateRoot< TKey >` implement the `IFullAuditedObject` interface.

All these base classes also have non-generic versions to take `AuditedEntity` and `FullAuditedAggregateRoot` to support the composite primary keys.

All these base classes also have `...WithUser` pairs, like `FullAuditedAggregateRootWithUser< TUser >` and `FullAuditedAggregateRootWithUser< TKey, TUser >`. This makes possible to add a navigation property to your user entity. However, it is not a good practice to add navigation properties between aggregate roots, so this usage is not suggested (unless you are using an ORM, like EF Core, that well supports this scenario and you really need it - otherwise remember that this approach doesn't work for NoSQL databases like MongoDB where you must truly implement the aggregate pattern). Also, if you add navigation properties to the AppUser

class that comes with the startup template, consider to handle (ignore/map) it on the `migration dbcontext` (see [[the EF Core migration document](#)](Entity-Framework-Core-Migrations.md)).

Caching Entities

ABP Framework provides a [[Distributed Entity Cache System](#)](Entity-Cache.md) for caching entities. It is useful if you want to use caching for quicker access to the entity rather than repeatedly querying it from the database.

It's designed as read-only and automatically invalidates a cached entity if the entity is updated or deleted.

> See the [[Entity Cache](#)](Entity-Cache.md) documentation for more information.

Versioning Entities

ABP defines the '[IHasEntityVersion](#)' interface for automatic versioning of your entities. It only provides a single '[EntityVersion](#)' property, as shown in the following code block:

```
```csharp
public interface IHasEntityVersion
{
 int EntityVersion { get; }
}
...```

```

If you implement the '[IHasEntityVersion](#)' interface, ABP automatically increases the '[EntityVersion](#)' value whenever you update your entity. The initial '[EntityVersion](#)' value will be '`0`', when you first create an entity and save to the database.

> ABP can not increase the version if you directly execute SQL '[UPDATE](#)' commands in the database. It is your responsibility to increase the '[EntityVersion](#)' value in that case. Also, if you are using the aggregate pattern and change sub-collections of an aggregate root, it is your responsibility if you want to increase the version of the aggregate root object.

## ## Extra Properties

ABP defines the '[IHasExtraProperties](#)' interface that can be implemented by an entity to be able to dynamically set and get properties for the entity. '[AggregateRoot](#)' base class already implements the '[IHasExtraProperties](#)' interface. If you've derived from this class (or one of the related audit class defined above), you can directly use the API.

### ### GetProperty & SetProperty Extension Methods

These extension methods are the recommended way to get and set data for an entity. Example:

```
```csharp
public class ExtraPropertiesDemoService : ITransientDependency
{
    private readonly IIIdentityUserRepository _identityUserRepository;
...
```

```

```

 public ExtraPropertiesDemoService(IIdentityUserRepository
identityUserRepository)
{
 _identityUserRepository = identityUserRepository;
}

public async Task SetTitle(Guid userId, string title)
{
 var user = await _identityUserRepository.GetAsync(userId);

 //SET A PROPERTY
 user SetProperty("Title", title);

 await _identityUserRepository.UpdateAsync(user);
}

public async Task<string> GetTitle(Guid userId)
{
 var user = await _identityUserRepository.GetAsync(userId);

 //GET A PROPERTY
 return user.GetProperty<string>("Title");
}
```

```

* Property's **value is object** and can be any type of object (string, int, bool... etc).
* `GetProperty` returns `null` if given property was not set before.
* You can store more than one property at the same time by using different **property names** (like `Title` here).

It would be a good practice to **define a constant** for the property name to prevent typo errors. It would be even a better practice to **define extension methods** to take the advantage of the intellisense. Example:

```

```csharp
public static class IdentityUserExtensions
{
 private const string TitlePropertyName = "Title";

 public static void SetTitle(this IdentityUser user, string title)
 {
 user SetProperty(TitlePropertyName, title);
 }

 public static string GetTitle(this IdentityUser user)
 {
 return user.GetProperty<string>(TitlePropertyName);
 }
```

```

Then you can directly use `user.SetTitle("...")` and `user.GetTitle()` for an `IdentityUser` object.

HasProperty & RemoveProperty Extension Methods

- * `'HasProperty'` is used to check if the object has a property set before.
- * `'RemoveProperty'` is used to remove a property from the object. You can use this instead of setting a `'null'` value.

How it is Implemented?

`'IHasExtraProperties'` interface requires to define a `'Dictionary<string, object>'` property, named `'ExtraProperties'`, for the implemented class.

So, you can directly use the `'ExtraProperties'` property to use the dictionary API, if you like. However, `'SetProperty'` and `'GetProperty'` methods are the recommended ways since they also check for `'null'`'s.

How is it Stored?

The way to store this dictionary in the database depends on the database provider you're using.

- * For [Entity Framework Core](Entity-Framework-Core.md), here are two type of configurations:
 - * By default, it is stored in a single `'ExtraProperties'` field as a `'JSON'` string (that means all extra properties stored in a single database table field). Serializing to `'JSON'` and deserializing from the `'JSON'` are automatically done by the ABP Framework using the [value conversions](<https://docs.microsoft.com/en-us/ef/core/modeling/value-conversions>) system of the EF Core.
 - * If you want, you can use the `'ObjectExtensionManager'` to define a separate table field for a desired extra property. Properties those are not configured through the `'ObjectExtensionManager'` will continue to use a single `'JSON'` field as described above. This feature is especially useful when you are using a pre-built [application module](Modules/Index.md) and want to [extend its entities](Customizing-Application-Modules-Extending-Entities.md). See the [EF Core integration document](Entity-Framework-Core.md) to learn how to use the `'ObjectExtensionManager'`.
- * For [MongoDB](MongoDB.md), it is stored as a **regular field**, since MongoDB naturally supports this kind of [extra elements](<https://mongodb.github.io/mongo-csharp-driver/1.11/serialization/#supporting-extra-elements>) system.

Discussion for the Extra Properties

Extra Properties system is especially useful if you are using a **re-usable module** that defines an entity inside and you want to get/set some data related to this entity in an easy way.

You typically **don't need** to use this system for your own entities, because it has the following drawbacks:

- * It is **not fully type safe** since it works with strings as property names.
- * It is **not easy to [auto map](Object-To-Object-Mapping.md)** these properties from/to other objects.

Extra Properties Behind Entities

`'IHasExtraProperties'` is not restricted to be used with entities. You can implement this interface for any kind of class and use the `'GetProperty'`, `'SetProperty'` and other related methods.

See Also

- * [Best practice guide to design the entities](Best-Practices/Entities.md)

8.2.2.2 Value Objects

Value Objects

- > An object that represents a descriptive aspect of the domain with no conceptual identity is called a VALUE OBJECT.
- >
- > (Eric Evans)

Two [Entities](Entities.md) with the same properties but with different `Id`'s are considered as different entities. However, Value Objects have no `Id`'s and they are considered as equals if they have the same property values.

The ValueObject Class

'ValueObject' is an abstract class that can be inherited to create a Value Object class.

****Example: An Address class****

```
```csharp
public class Address : ValueObject
{
 public Guid CityId { get; private set; }

 public string Street { get; private set; }

 public int Number { get; private set; }

 private Address()
 {

 }

 public Address(
 Guid cityId,
 string street,
 int number)
 {
 CityId = cityId;
 Street = street;
 Number = number;
 }

 protected override IEnumerable<object> GetAtomicValues()
 {
 yield return Street;
 yield return CityId;
 yield return Number;
 }
}
```

....

\* A Value Object class must implement the `GetAtomicValues()` method to return the primitive values.

### ### `ValueEquals`

`ValueObject.ValueEquals(...)` method is used to check if two Value Objects are equals.

\*\*Example: Check if two addresses are equals\*\*

```csharp

```
Address address1 = ...
```

```
Address address2 = ...
```

```
if (address1.ValueEquals(address2)) //Check equality
{
```

```
    ...
}
```

```
....
```

Best Practices

Here are some best practices when using Value Objects:

- Design a value object as **immutable** (like the Address above) if there is not a good reason for designing it as mutable.
- The properties that make up a Value Object should form a conceptual whole. For example, CityId, Street and Number shouldn't be separate properties of a Person entity. This also makes the Person entity simpler.

See Also

* [Entities](Entities.md)

8.2.2.3 *Repositories*

Repositories

"*Mediates between the domain and data mapping layers using a collection-like interface for accessing domain objects*"(Martin Fowler).

Repositories, in practice, are used to perform database operations for domain objects (see [Entities](Entities.md)). Generally, a separated repository is used for each **aggregate root** or entity.

Generic Repositories

ABP can provide a **default generic repository** for each aggregate root or entity. You can [inject](Dependency-Injection.md) ` IRepository< TEntity, TKey >` into your service and perform standard **CRUD** operations.

> Database provider layer should be properly configured to be able to use the default generic repositories. It is **already done** if you've created your project using the startup templates. If not, refer to the database provider

documents ([EF Core](Entity-Framework-Core.md) / [MongoDB](MongoDB.md)) to configure it.

Example usage of a default generic repository:

```
```C#
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace Demo
{
 public class PersonAppService : ApplicationService
 {
 private readonly IRepository<Person, Guid> _personRepository;

 public PersonAppService(IRepository<Person, Guid> personRepository)
 {
 _personRepository = personRepository;
 }

 public async Task CreateAsync(CreatePersonDto input)
 {
 var person = new Person(input.Name);

 await _personRepository.InsertAsync(person);
 }

 public async Task<int> GetCountAsync(string filter)
 {
 return await _personRepository.CountAsync(p =>
p.Name.Contains(filter));
 }
 }
}...```

```

In this example;

- \* `PersonAppService` simply injects `IRepository<Person, Guid>` in its constructor.
- \* `CreateAsync` method uses `InsertAsync` to save the new entity.
- \* `GetCountAsync` method gets a filtered count of all people in the database.

### ### Standard Repository Methods

Generic Repositories provides some standard CRUD features out of the box:

- \* `GetAsync`: Returns a single entity by its `Id` or a predicate (lambda expression).
  - \* Throws `EntityNotFoundException` if the requested entity was not found.
  - \* Throws `InvalidOperationException` if there are multiple entities with given predicate.
- \* `FindAsync`: Returns a single entity by its `Id` or a predicate (lambda expression).
  - \* Returns `null` if the requested entity was not found.

```

 * Throws `InvalidOperationException` if there are multiple entities with
given predicate.
 * `InsertAsync`: Inserts a new entity to the database.
 * `UpdateAsync`: Updates an existing entity in the database.
 * `DeleteAsync`: Deletes the given entity from database.
 * This method has an overload that takes a predicate (lambda expression) to
delete multiple entities satisfies the given condition.
 * `GetListAsync`: Returns the list of all entities in the database.
 * `GetPagedListAsync`: Returns a limited list of entities. Gets `skipCount`,
`maxResultCount` and `sorting` parameters.
 * `GetCountAsync`: Gets count of all entities in the database.

```

There are overloads of these methods.

- \* Provides `UpdateAsync` and `DeleteAsync` methods to update or delete an entity by entity object or it's id.
- \* Provides `DeleteAsync` method to delete multiple entities by a filter.

### ### Querying / LINQ over the Repositories

Repositories provide the `GetQueryableAsync()` method that returns an `IQueryable< TEntity >` object. You can use this object to perform LINQ queries on the entities in the database.

**\*\*Example: Use LINQ with the repositories\*\***

```

```csharp
using System;
using System.Linq;
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace Demo
{
    public class PersonAppService : ApplicationService
    {
        private readonly IRepository<Person, Guid> _personRepository;

        public PersonAppService(IRepository<Person, Guid> personRepository)
        {
            _personRepository = personRepository;
        }

        public async Task<List<PersonDto>> GetListAsync(string filter)
        {
            //Obtain the IQueryable<Person>
            IQueryable<Person> queryable = await
            _personRepository.GetQueryableAsync();

            //Create a query
            var query = from person in queryable
                       where person.Name == filter
                       orderby person.Name
                       select person;

            //Execute the query to get list of people
        }
    }
}
```

```

```

 var people = query.ToList();

 //Convert to DTO and return to the client
 return people.Select(p => new PersonDto {Name =
p.Name}).ToList();
 }
}
```

```

You could also use the LINQ extension methods:

```

```csharp
public async Task<List<PersonDto>> GetListAsync(string filter)
{
 //Obtain the IQueryable<Person>
 IQueryable<Person> queryable = await
_personRepository.GetQueryableAsync();

 //Execute a query
 var people = queryable
 .Where(p => p.Name.Contains(filter))
 .OrderBy(p => p.Name)
 .ToList();

 //Convert to DTO and return to the client
 return people.Select(p => new PersonDto {Name = p.Name}).ToList();
}
```

```

Any standard LINQ method can be used over the `IQueryable` returned from the repository.

> This sample uses `ToListAsync` method, but it is **strongly suggested to use the asynchronous methods** to perform database queries, like `ToListAsync()` for this example. See the **`IQueryable` & Async Operations** section to learn how you can do it.

> **Exposing `IQueryable` to outside of a repository** class may leak your data access logic to the application layer. If you want to strictly follow the **layered architecture** principles, you can consider to implement a custom repository class and wrap your data access logic inside your repository class. You can see the ***Custom Repositories*** section to learn how to create custom repository classes for your application.

Bulk Operations

There are some methods to perform bulk operations in the database;

- * `InsertManyAsync`
- * `UpdateManyAsync`
- * `DeleteManyAsync`

These methods work with multiple entities and can take advantage of bulk operations if supported by the underlying database provider.

> Optimistic concurrency control may not be possible when you use `UpdateManyAsync` and `DeleteManyAsync` methods.

Soft / Hard Delete

`DeleteAsync` method of the repository doesn't delete the entity if the entity is a **soft-delete** entity (that implements `ISoftDelete`). Soft-delete entities are marked as "deleted" in the database. Data Filter system ensures that the soft deleted entities are not retrieved from database normally.

If your entity is a soft-delete entity, you can use the `HardDeleteAsync` method to physically delete the entity from database in case of you need it.

> See the [Data Filtering](Data-Filtering.md) documentation for more about soft-delete.

Delete Direct

`DeleteDirectAsync` method of the repository deletes all entities those fit to the given predicate. It directly deletes entities from database, without fetching them.

Some features (like soft-delete, multi-tenancy and audit logging) won't work, so use this method carefully when you need it. Use the `DeleteAsync` method if you need to these features.

> Currently only [EF Core supports it](<https://learn.microsoft.com/en-us/ef/core/what-is-new/ef-core-7.0/whatsnew#basic-executedelete-examples>), For the ORMs doesn't support direct delete, we will fallback to the existing `DeleteAsync` method.

Ensure Entities Exists

The `EnsureExistsAsync` extension method accepts entity id or entities query expression to ensure entities exist, otherwise, it will throw `EntityNotFoundException` .

Other Generic Repository Types

Standard ` IRepository< TEntity, TKey >` interface exposes the standard ` IQueryable< TEntity >` and you can freely query using the standard LINQ methods. This is fine for most of the applications. However, some ORM providers or database systems may not support standard ` IQueryable` interface. If you want to use such providers, you can't rely on the ` IQueryable` .

Basic Repositories

ABP provides `IBasicRepository< TEntity, TPrimaryKey >` and `IBasicRepository< TEntity >` interfaces to support such scenarios. You can extend these interfaces (and optionally derive from `BasicRepositoryBase`) to create custom repositories for your entities.

Depending on `IBasicRepository` but not depending on ` IRepository` has an advantage to make possible to work with all data sources even if they don't support ` IQueryable` .

Major vendors, like Entity Framework, NHibernate or MongoDB already support ` IQueryable` . So, working with ` IRepository` is the **suggested** way for

typical applications. But reusable module developers may consider '[IBasicRepository](#)' to support a wider range of data sources.

Read Only Repositories

There are also '[IReadOnlyRepository< TEntity, TKey >](#)' and '[IReadOnlyBasicRepository< TEntity, TKey >](#)' interfaces for who only want to depend on querying capabilities of the repositories.

The '[IReadOnlyRepository< TEntity, TKey >](#)' derives the '[IReadOnlyBasicRepository< TEntity, TKey >](#)' and provides the following properties and methods as well:

Properties:

'[AsyncExecuter](#)' : a service that is used to execute an '[IQueryable< T >](#)' object asynchronously **without depending on the actual database provider**.

Methods:

- '[GetListAsync\(\)](#)'
- '[GetQueryableAsync\(\)](#)'
- '[WithDetails\(\)](#)' 1 overload
- '[WithDetailsAsync\(\)](#)' 1 overload

Where as the '[IReadOnlyBasicRepository< TEntity, TKey >](#)' provides the following methods:

- '[GetCountAsync\(\)](#)'
- '[GetListAsync\(\)](#)'
- '[GetPagedListAsync\(\)](#)'

They can all be seen as below:

![generic-repositories](images/generic-repositories.png)

Read Only Repositories behavior in Entity Framework Core

Entity Framework Core read-only repository implementation uses [[EF Core's No-Tracking feature](#)](<https://learn.microsoft.com/en-us/ef/core/querying/tracking#no-tracking-queries>). That means the entities returned from the repository will not be tracked by the EF Core [[change tracker](#)](<https://learn.microsoft.com/en-us/ef/core/change-tracking/>), because it is expected that you won't update entities queried from a read-only repository. If you need to track the entities, you can still use [[AsTracking\(\)](#)](<https://learn.microsoft.com/en-us/dotnet/api/microsoft.entityframeworkcore.entityframeworkqueryableextensions.astracking>) extension method.

> This behavior works only if the repository object is injected with one of the read-only repository interfaces ('[IReadOnlyRepository< ... >](#)' or '[IReadOnlyBasicRepository< ... >](#)'). It won't work if you have injected a standard repository (e.g. ' [IRepository< ... >](#)') then casted it to a read-only repository interface.

Generic Repository without a Primary Key

If your entity does not have an Id primary key (it may have a composite primary key for instance) then you cannot use the ` IRepository< TEntity, TKey >` (or basic/readonly versions) defined above. In that case, you can inject and use ` IRepository< TEntity >` for your entity.

> ` IRepository< TEntity >` has a few missing methods those normally works with the ` Id ` property of an entity. Because of the entity has no ` Id ` property in that case, these methods are not available. One example is the ` Get ` method that gets an id and returns the entity with given id. However, you can still use ` IQueryable< TEntity >` features to query entities by standard LINQ methods.

Custom Repositories

Default generic repositories will be sufficient for most cases. However, you may need to create a custom repository class for your entity.

Custom Repository Example

ABP does not force you to implement any interface or inherit from any base class for a repository. It can be just a simple POCO class. However, it's suggested to inherit existing repository interface and classes to make your work easier and get the standard methods out of the box.

Custom Repository Interface

First, define an interface in your domain layer:

```
```c#
public interface IPersonRepository : IRepository<Person, Guid>
{
 Task<Person> FindByNameAsync(string name);
}
...```

```

This interface extends ` IRepository< Person, Guid >` to take advantage of pre-built repository functionality.

#### #### Custom Repository Implementation

A custom repository is tightly coupled to the data access tool type you are using. In this example, we will use Entity Framework Core:

```
```C#
public class PersonRepository : EfCoreRepository<MyDbContext, Person, Guid>, IPersonRepository
{
    public PersonRepository(IDbContextProvider<TestAppDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }

    public async Task<Person> FindByNameAsync(string name)
    {
        var dbContext = await GetDbContextAsync();
        return await dbContext.Set<Person>()
...```

```

```
        .Where(p => p.Name == name)
        .FirstOrDefaultAsync();
    }
}
```

You can directly access the data access provider (``DbContext`` in this case) to perform operations.

> See [EF Core](Entity-Framework-Core.md) or [MongoDb](MongoDB.md) document for more info about the custom repositories.

`IQueryable & Async Operations`

`' IRepository'` provides `'GetQueryableAsync()'` to obtain an `' IQueryable'`, that means you can **directly use LINQ extension methods** on it, as shown in the example of the "**Querying / LINQ over the Repositories**" section above.

Example: Using the `'Where(...)` and the `'ToListAsync()'` extension methods

```
```csharp
var queryable = await _personRepository.GetQueryableAsync();
var people = queryable
 .Where(p => p.Name.Contains(nameFilter))
 .ToList();
```
```

`' .ToListAsync(), ' .CountAsync()'`... are standard extension methods defined in the `' System.Linq'` namespace ([see all](https://docs.microsoft.com/en-us/dotnet/api/system.linq.queryable)).

You normally want to use `' .ToListAsync()`, `' .CountAsync()'`... instead, to be able to write a **truly async code**.

However, you see that you can't use all the async extension methods in your application or domain layer when you create a new project using the standard [application startup template](Startup-Templates/Application.md), because;

- * These async methods **are not standard LINQ methods** and they are defined in the [Microsoft.EntityFrameworkCore](https://www.nuget.org/packages/Microsoft.EntityFrameworkCore) NuGet package.
- * The standard template **doesn't have a reference** to the EF Core package from the domain and application layers, to be independent from the database provider.

Based on your requirements and development model, you have the following options to be able to use the async methods.

> Using async methods is strongly suggested! Don't use sync LINQ methods while executing database queries to be able to develop a scalable application.

`Option-1: Reference to the Database Provider Package`

The easiest solution is to directly add the EF Core package from the project you want to use these async methods.

> Add the [Volo.Abp.EntityFrameworkCore](<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore>) NuGet package to your project, which indirectly reference to the EF Core package. This ensures that you use the correct version of the EF Core compatible to the rest of your application.

When you add the NuGet package to your project, you can take full power of the EF Core extension methods.

****Example: Directly using the `ToListAsync()` after adding the EF Core package****

```
```csharp
var queryable = await _personRepository.GetQueryableAsync();
var people = queryable
 .Where(p => p.Name.Contains(nameFilter))
 .ToListAsync();
....
```

This method is suggested;

\* If you are developing an application and you \*\*don't plan to change\*\* EF Core in the future, or you can \*\*tolerate\*\* it if you need to change later. We believe that's reasonable if you are developing a final application.

#### #### MongoDB Case

If you are using [MongoDB](MongoDB.md), you need to add the [Volo.Abp.MongoDB](<https://www.nuget.org/packages/Volo.Abp.MongoDB>) NuGet package to your project. Even in this case, you can't directly use async LINQ extensions (like `ToListAsync`) because MongoDB doesn't provide async extension methods for `IQueryable<T>`, but provides for `IMongoQueryable<T>`. You need to cast the query to `IMongoQueryable<T>` first to be able to use the async extension methods.

**\*\*Example: Cast `IQueryable<T>` to `IMongoQueryable<T>` and use `ToListAsync()`\*\***

```
```csharp
var queryable = await _personRepository.GetQueryableAsync();
var people = ((IMongoQueryable<Person>) queryable
    .Where(p => p.Name.Contains(nameFilter)))
    .ToListAsync();
....
```

Option-2: Use the IRepository Async Extension Methods

ABP Framework provides async extension methods for the repositories, just similar to async LINQ extension methods.

****Example: Use `CountAsync` and `FirstOrDefaultAsync` methods on the repositories****

```
```csharp
var countAll = await _personRepository
 .CountAsync();

var count = await _personRepository
```

```

 .CountAsync(x => x.Name.StartsWith("A"));

var book1984 = await _bookRepository
 .FirstOrDefaultAsync(x => x.Name == "John");
...

```

The standard LINQ extension methods are supported: \**AllAsync*, *AnyAsync*, *AverageAsync*, *ContainsAsync*, *CountAsync*, *FirstAsync*, *FirstOrDefaultAsync*, *LastAsync*, *LastOrDefaultAsync*, *LongCountAsync*, *MaxAsync*, *MinAsync*, *SingleAsync*, *SingleOrDefaultAsync*, *SumAsync*, *ToArrayListAsync*\*.

This approach still \*\*has a limitation\*\*. You need to call the extension method directly on the repository object. For example, the below usage is \*\*not supported\*\*:

```

```csharp
var queryable = await _bookRepository.GetQueryableAsync();
var count = await queryable.Where(x => x.Name.Contains("A")).CountAsync();
```

```

This is because the '`CountAsync()`' method in this example is called on a '`IQueryable`' interface, not on the repository object. See the other options for such cases.

This method is suggested \*\*wherever possible\*\*.

### ### Option-3: `IAsyncQueryableExecuter`

`'IAsyncQueryableExecuter'` is a service that is used to execute an '`IQueryable<T>`' object asynchronously \*\*without depending on the actual database provider\*\*.

\*\*Example: Inject & use the '`IAsyncQueryableExecuter.ToListAsync()`' method\*\*

```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Linq;

namespace AbpDemo
{
    public class ProductAppService : ApplicationService, IProductAppService
    {
        private readonly IRepository<Product, Guid> _productRepository;
        private readonly IAsyncQueryableExecuter _asyncExecuter;

        public ProductAppService(
            IRepository<Product, Guid> productRepository,
            IAsyncQueryableExecuter asyncExecuter)
        {
            _productRepository = productRepository;
            _asyncExecuter = asyncExecuter;
        }
    }
}

```

```

        public async Task<ListResultDto<ProductDto>> GetListAsync(string
name)
{
    //Obtain the IQueryable<T>
    var queryable = await _productRepository.GetQueryableAsync();

    //Create the query
    var query = queryable
        .Where(p => p.Name.Contains(name))
        .OrderBy(p => p.Name);

    //Run the query asynchronously
    List<Product> products = await _asyncExecuter.ToListAsync(query);

    //...
}
}
...

```

> `ApplicationService` and `DomainService` base classes already have `AsyncExecuter` properties pre-injected and usable without needing an explicit constructor injection.

ABP Framework executes the query asynchronously using the actual database provider's API. While that is not a usual way to execute a query, it is the best way to use the async API without depending on the database provider.

This method is suggested;

- * If you want to develop your application code **without depending** on the database provider.
- * If you are building a **reusable library** that doesn't have a database provider integration package, but needs to execute an `IQueryable<T>` object in some case.

For example, ABP Framework uses the `IAsyncQueryableExecuter` in the `CrudAppService` base class (see the [application services](Application-Services.md) document).

[### Option-4: Custom Repository Methods](#)

You can always create custom repository methods and use the database provider specific APIs, like async extension methods here. See [EF Core](Entity-Framework-Core.md) or [MongoDb](MongoDB.md) document for more info about the custom repositories.

This method is suggested;

- * If you want to **completely isolate** your domain & application layers from the database provider.
- * If you develop a **reusable [application module](Modules/Index.md)** and don't want to force to a specific database provider, which should be done as a [best practice](Best-Practices/Index.md).

8.2.2.4 Domain Services

```
# Domain Services
```

```
## Introduction
```

In a [Domain Driven Design](Domain-Driven-Design.md) (DDD) solution, the core business logic is generally implemented in aggregates ([entities](Entities.md)) and the Domain Services. Creating a Domain Service is especially needed when;

- * You implement a core domain logic that depends on some services (like repositories or other external services).
- * The logic you need to implement is related to more than one aggregate/entity, so it doesn't properly fit in any of the aggregates.

```
## ABP Domain Service Infrastructure
```

Domain Services are simple, stateless classes. While you don't have to derive from any service or interface, ABP Framework provides some useful base classes and conventions.

```
### DomainService & IDomainService
```

Either derive a Domain Service from the `DomainService` base class or directly implement the `IDomainService` interface.

****Example:** Create a Domain Service deriving from the `DomainService` base class.**

```
```csharp
using Volo.Abp.Domain.Services;

namespace MyProject.Issues
{
 public class IssueManager : DomainService
 {

 }
}
...```

```

When you do that;

- \* ABP Framework automatically registers the class to the Dependency Injection system with a Transient lifetime.
- \* You can directly use some common services as base properties, without needing to manually inject (e.g. [ILogger](Logging.md) and [IGuidGenerator](Guid-Generation.md)).

> It is suggested to name a Domain Service with a `Manager` or `Service` suffix. We typically use the `Manager` suffix as used in the sample above.

**\*\*Example:** Implement the domain logic of assigning an Issue to a User\*\*

```
```csharp
public class IssueManager : DomainService
{
    ...
}
```

```

{
    private readonly IRepository<Issue, Guid> _issueRepository;

    public IssueManager(IRepository<Issue, Guid> issueRepository)
    {
        _issueRepository = issueRepository;
    }

    public async Task AssignAsync(Issue issue, AppUser user)
    {
        var currentIssueCount = await _issueRepository
            .CountAsync(i => i.AssignedUserId == user.Id);

        //Implementing a core business validation
        if (currentIssueCount >= 3)
        {
            throw new IssueAssignmentException(user.UserName);
        }

        issue.AssignedUserId = user.Id;
    }
}
...

```

Issue is an [aggregate root](Entities.md) defined as shown below:

```

```csharp
public class Issue : AggregateRoot<Guid>
{
 public Guid? AssignedUserId { get; internal set; }

 //...
}
...

```

\* Making the setter `internal` ensures that it can not directly set in the upper layers and forces to always use the `IssueManager` to assign an `Issue` to a `User`.

### ### Using a Domain Service

A Domain Service is typically used in an [application service](Application-Services.md).

\*\*Example: Use the `IssueManager` to assign an Issue to a User\*\*

```

```csharp
using System;
using System.Threading.Tasks;
using MyProject.Users;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace MyProject.Issues
{
    public class IssueAppService : ApplicationService, IIssueAppService
    {
        private readonly IssueManager _issueManager;

```

```

private readonly IRepository<AppUser, Guid> _userRepository;
private readonly IRepository<Issue, Guid> _issueRepository;

public IssueAppService(
    IssueManager issueManager,
    IRepository<AppUser, Guid> userRepository,
    IRepository<Issue, Guid> issueRepository)
{
    _issueManager = issueManager;
    _userRepository = userRepository;
    _issueRepository = issueRepository;
}

public async Task AssignAsync(Guid id, Guid userId)
{
    var issue = await _issueRepository.GetAsync(id);
    var user = await _userRepository.GetAsync(userId);

    await _issueManager.AssignAsync(issue, user);
    await _issueRepository.UpdateAsync(issue);
}
}
...
}
...
}

```

Since the `IssueAppService` is in the Application Layer, it can't directly assign an issue to a user. So, it uses the `IssueManager`.

Application Services vs Domain Services

While both of [Application Services](Application-Services.md) and Domain Services implement the business rules, there are fundamental logical and formal differences;

- * Application Services implement the **use cases** of the application (user interactions in a typical web application), while Domain Services implement the **core, use case independent domain logic**.
- * Application Services get/return [Data Transfer Objects](Data-Transfer-Objects.md), Domain Service methods typically get and return the **domain objects** ([entities](Entities.md), [value objects](Value-Objects.md)).
- * Domain services are typically used by the Application Services or other Domain Services, while Application Services are used by the Presentation Layer or Client Applications.

Lifetime

Lifetime of Domain Services are [transient](<https://docs.abp.io/en/abp/latest/Dependency-Injection>) and they are automatically registered to the dependency injection system.

8.2.2.5 Specifications

Specifications

Specification Pattern is used to define **named, reusable, combinable and testable filters** for entities and other business objects.

> A Specification is a part of the Domain Layer.

Installation

> This package is **already installed** when you use the startup templates. So, most of the times you don't need to manually install it.

Install the [Volo.Abp.Specifications](<https://abp.io/package-detail/Volo.Abp.Specifications>) package to your project. You can use the [ABP CLI](CLI.md) `*add-package*` command in a command line terminal when the current folder is the root folder of your project (`'csproj'`):

```
```bash
abp add-package Volo.Abp.Specifications
````
```

Defining the Specifications

Assume that you've a Customer entity as defined below:

```
```csharp
using System;
using Volo.Abp.Domain.Entities;

namespace MyProject
{
 public class Customer : AggregateRoot<Guid>
 {
 public string Name { get; set; }

 public byte Age { get; set; }

 public long Balance { get; set; }

 public string Location { get; set; }
 }
}
````
```

You can create a new Specification class derived from the `'Specification<Customer>'`.

Example: A specification to select the customers with 18+ age:

```
```csharp
using System;
using System.Linq.Expressions;
using Volo.Abp.Specifications;

namespace MyProject
{
 public class Age18PlusCustomerSpecification : Specification<Customer>
 {
 public override Expression<Func<Customer, bool>> ToExpression()
 {
 return c => c.Age >= 18;
 }
 }
}
````
```

```
}...
```

You simply define a lambda [[Expression](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions)](<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/lambda-expressions>) to define a specification.

> Instead, you can directly implement the `ISpecification<T>` interface, but the `Specification<T>` base class much simplifies it.

Using the Specifications

There are two common use cases of the specifications.

IsSatisfiedBy

`IsSatisfiedBy` method can be used to check if a single object satisfies the specification.

Example: Throw exception if the customer doesn't satisfy the age specification

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;

namespace MyProject
{
 public class CustomerService : ITransientDependency
 {
 public async Task BuyAlcohol(Customer customer)
 {
 if (!new
Age18PlusCustomerSpecification().IsSatisfiedBy(customer))
 {
 throw new Exception(
 "This customer doesn't satisfy the Age specification!");
 }
 }

 //TODO...
 }
}
```

### ### ToExpression & Repositories

`ToExpression()` method can be used to use the specification as Expression. In this way, you can use a specification to \*\*filter entities while querying from the database\*\*.

```
```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
```

```

using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Domain.Services;

namespace MyProject
{
    public class CustomerManager : DomainService, ITransientDependency
    {
        private readonly IRepository<Customer, Guid> _customerRepository;

        public CustomerManager(IRepository<Customer, Guid>
customerRepository)
        {
            _customerRepository = customerRepository;
        }

        public async Task<List<Customer>> GetCustomersCanBuyAlcohol()
        {
            var queryable = await _customerRepository.GetQueryableAsync();
            var query = queryable.Where(
                new Age18PlusCustomerSpecification().ToExpression()
            );

            return await AsyncExecuter.ToListAsync(query);
        }
    }
}
```

```

> Specifications are correctly translated to SQL/Database queries and executed efficiently in the DBMS side. While it is not related to the Specifications, see the [Repositories](Repositories.md) document if you want to know more about the 'AsyncExecuter'.

Actually, using the 'ToExpression()' method is not necessary since the specifications are automatically casted to Expressions. This would also work:

```

```csharp
var queryable = await _customerRepository.GetQueryableAsync();
var query = queryable.Where(
    new Age18PlusCustomerSpecification()
);
```

```

#### ## Composing the Specifications

One powerful feature of the specifications is that they are composable with 'And', 'Or', 'Not' and 'AndNot' extension methods.

Assume that you have another specification as defined below:

```

```csharp
using System;
using System.Linq.Expressions;
using Volo.Abp.Specifications;

namespace MyProject
{

```

```

public class PremiumCustomerSpecification : Specification<Customer>
{
    public override Expression<Func<Customer, bool>> ToExpression()
    {
        return (customer) => (customer.Balance >= 100000);
    }
}
...

```

You can combine the `PremiumCustomerSpecification` with the `Age18PlusCustomerSpecification` to query the count of premium adult customers as shown below:

```

```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Domain.Services;
using Volo.Abp.Specifications;

namespace MyProject
{
 public class CustomerManager : DomainService, ITransientDependency
 {
 private readonly IRepository<Customer, Guid> _customerRepository;

 public CustomerManager(IRepository<Customer, Guid>
customerRepository)
 {
 _customerRepository = customerRepository;
 }

 public async Task<int> GetAdultPremiumCustomerCountAsync()
 {
 return await _customerRepository.CountAsync(
 new Age18PlusCustomerSpecification()
 .And(new PremiumCustomerSpecification()).ToExpression()
);
 }
 }
}
```

```

If you want to make this combination another reusable specification, you can create such a combination specification class deriving from the `AndSpecification`:

```

```csharp
using Volo.Abp.Specifications;

namespace MyProject
{
 public class AdultPremiumCustomerSpecification :
AndSpecification<Customer>
 {
 public AdultPremiumCustomerSpecification()
 {
 }
 }
}
```

```

```

        : base(new Age18PlusCustomerSpecification(),
              new PremiumCustomerSpecification())
    }
}
}...
}

```

Now, you can re-write the `GetAdultPremiumCustomerCountAsync` method as shown below:

```

```csharp
public async Task<int> GetAdultPremiumCustomerCountAsync()
{
 return await _customerRepository.CountAsync(
 new AdultPremiumCustomerSpecification()
);
}
```

```

> You see the power of the specifications with these samples. If you change the `PremiumCustomerSpecification` later, say change the balance from `100.000` to `200.000`, all the queries and combined specifications will be effected by the change. This is a good way to reduce code duplication!

Discussions

While the specification pattern is older than C# lambda expressions, it's generally compared to expressions. Some developers may think it's not needed anymore and we can directly pass expressions to a repository or to a domain service as shown below:

```

```csharp
var count = await _customerRepository.CountAsync(c => c.Balance > 100000 &&
c.Age >= 18);
```

```

Since ABP's [Repository](Repositories.md) supports Expressions, this is a completely valid use. You don't have to define or use any specification in your application and you can go with expressions.

So, what's the point of a specification? Why and when should we consider to use them?

When To Use?

Some benefits of using specifications:

- **Reusable**: Imagine that you need the Premium Customer filter in many places in your code base. If you go with expressions and do not create a specification, what happens if you later change the "Premium Customer" definition? Say you want to change the minimum balance from \$100,000 to \$250,000 and add another condition to be a customer older than 3 years. If you'd used a specification, you just change a single class. If you repeated (copy/pasted) the same expression everywhere, you need to change all of them.
- **Composable**: You can combine multiple specifications to create new specifications. This is another type of reusability.

- **Named**: `PremiumCustomerSpecification` better explains the intent rather than a complex expression. So, if you have an expression that is meaningful in your business, consider using specifications.
- **Testable**: A specification is a separately (and easily) testable object.

When To Not Use?

- **Non business expressions**: Do not use specifications for non business-related expressions and operations.
- **Reporting**: If you are just creating a report, do not create specifications, but directly use `IQueryable` & LINQ expressions. You can even use plain SQL, views or another tool for reporting. DDD does not necessarily care about reporting, so the way you query the underlying data store can be important from a performance perspective.

8.2.3 Application Layer

8.2.3.1 Application Services

Application Services

Application services are used to implement the **use cases** of an application. They are used to **expose domain logic to the presentation layer**.

An Application Service is called from the presentation layer (optionally) with a **DTO ([Data Transfer Object])(Data-Transfer-Objects.md)** as the parameter. It uses domain objects to **perform some specific business logic** and (optionally) returns a DTO back to the presentation layer. Thus, the presentation layer is completely **isolated** from domain layer.

Example

Book Entity

Assume that you have a `Book` entity (actually, an aggregate root) defined as shown below:

```
```csharp
public class Book : AggregateRoot<Guid>
{
 public const int MaxNameLength = 128;

 public virtual string Name { get; protected set; }

 public virtual BookType Type { get; set; }

 public virtual float? Price { get; set; }

 protected Book()
 {

 }

 public Book(Guid id, [NotNull] string name, BookType type, float? price =
 0)
 {
```

```

 Id = id;
 Name = CheckName(name);
 Type = type;
 Price = price;
 }

 public virtual void ChangeName([NotNull] string name)
 {
 Name = CheckName(name);
 }

 private static string CheckName(string name)
 {
 if (string.IsNullOrWhiteSpace(name))
 {
 throw new ArgumentException(
 $"name can not be empty or white space!");
 }

 if (name.Length > MaxNameLength)
 {
 throw new ArgumentException(
 $"name can not be longer than {MaxNameLength} chars!");
 }

 return name;
 }
}

```
* `Book` entity has a `MaxNameLength` that defines the maximum length of the `Name` property.
* `Book` constructor and `ChangeName` method to ensure that the `Name` is always a valid value. Notice that `Name`'s setter is not `public`.

> ABP does not force you to design your entities like that. It just can have public get/set for all properties. It's your decision to fully implement DDD practices.
```

IBookAppService Interface

In ABP, an application service should implement the `IApplicationService` interface. It's good to create an interface for each application service:

```

```csharp
public interface IBookAppService : IApplicationService
{
 Task CreateAsync(CreateBookDto input);
}
```

```

A Create method will be implemented as the example. `CreateBookDto` is defined like that:

```

```csharp
public class CreateBookDto
{
 [Required]

```

```

[StringLength(Book.MaxNameLength)]
public string Name { get; set; }

public BookType Type { get; set; }
public float? Price { get; set; }
}...

```

> See [data transfer objects document](Data-Transfer-Objects.md) for more about DTOs.

```

BookAppService (Implementation)

```csharp
public class BookAppService : ApplicationService, IBookAppService
{
    private readonly IRepository<Book, Guid> _bookRepository;

    public BookAppService(IRepository<Book, Guid> bookRepository)
    {
        _bookRepository = bookRepository;
    }

    public async Task CreateAsync(CreateBookDto input)
    {
        var book = new Book(
            GuidGenerator.Create(),
            input.Name,
            input.Type,
            input.Price
        );

        await _bookRepository.InsertAsync(book);
    }
}...

```

- * `BookAppService` inherits from the `ApplicationService` base class. It's not required, but the `ApplicationService` class provides helpful properties for common application service requirements like `GuidGenerator` used in this service. If we didn't inherit from it, we would need to inject the `IGuidGenerator` service manually (see [guid generation](Guid-Generation.md) document).
- * `BookAppService` implements the `IBookAppService` as expected.
- * `BookAppService` [injects](Dependency-Injection.md) ` IRepository<Book, Guid>` (see [repositories](Repositories.md)) and uses it inside the `CreateAsync` method to insert a new entity to the database.
- * `CreateAsync` uses the constructor of the `Book` entity to create a new book from the properties of given `input`.

Data Transfer Objects

Application services get and return DTOs instead of entities. ABP does not force this rule. However, exposing entities to the presentation layer (or to remote clients) has significant problems and is not suggested.

See the [DTO documentation](Data-Transfer-Objects.md) for more.

Object to Object Mapping

The `CreateAsync` method above manually creates a `Book` entity from given `CreateBookDto` object, because the `Book` entity enforces it (we designed it like that).

However, in many cases, it's very practical to use **auto object mapping** to set properties of an object from a similar object. ABP provides an [object to object mapping](Object-To-Object-Mapping.md) infrastructure to make this even easier.

Object to object mapping provides abstractions and it is implemented by the [AutoMapper](https://automapper.org/) library by default.

Let's create another method to get a book. First, define the method in the `IBookAppService` interface:

```
```csharp
public interface IBookAppService : IApplicationService
{
 Task CreateAsync(CreateBookDto input);

 Task<BookDto> GetAsync(Guid id); //New method
}
````
```

`BookDto` is a simple [DTO](Data-Transfer-Objects.md) class defined as below:

```
```csharp
public class BookDto
{
 public Guid Id { get; set; }

 public string Name { get; set; }

 public BookType Type { get; set; }

 public float? Price { get; set; }
}
````
```

AutoMapper requires to create a mapping [profile class](https://docs.automapper.org/en/stable/Configuration.html#profile-instances). Example:

```
```csharp
public class MyProfile : Profile
{
 public MyProfile()
 {
 CreateMap<Book, BookDto>();
 }
}
````
```

You should then register profiles using the `AbpAutoMapperOptions`:

```

```csharp
[DependsOn(typeof(AbpAutoMapperModule))]
public class MyModule : AbpModule
{
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 Configure<AbpAutoMapperOptions>(options =>
 {
 //Add all mappings defined in the assembly of the MyModule class
 options.AddMaps<MyModule>();
 });
 }
}
```

```

``AddMaps`` registers all profile classes defined in the assembly of the given class, typically your module class. It also registers for the [attribute mapping](<https://docs.automapper.org/en/stable/Attribute-mapping.html>).

Then you can implement the `GetAsync` method as shown below:

```

```csharp
public async Task<BookDto> GetAsync(Guid id)
{
 var book = await _bookRepository.GetAsync(id);
 return ObjectMapper.Map<Book, BookDto>(book);
}
```

```

See the [object to object mapping document](Object-To-Object-Mapping.md) for more.

Validation

Inputs of application service methods are automatically validated (like ASP.NET Core controller actions). You can use the standard data annotation attributes or a custom validation method to perform the validation. ABP also ensures that the input is not null.

See the [validation document](Validation.md) for more.

Authorization

It's possible to use declarative and imperative authorization for application service methods.

See the [authorization document](Authorization.md) for more.

CRUD Application Services

If you need to create a simple **CRUD application service** which has Create, Update, Delete and Get methods, you can use ABP's **base classes** to easily build your services. You can inherit from the `CrudAppService`.

Example

Create an `IBookAppService` interface inheriting from the `ICrudAppService` interface.

```
```csharp
public interface IBookAppService :
 ICrudAppService< //Defines CRUD methods
 BookDto, //Used to show books
 Guid, //Primary key of the book entity
 PagedAndSortedResultRequestDto, //Used for paging/sorting on getting
 a list of books
 CreateUpdateBookDto, //Used to create a new book
 CreateUpdateBookDto> //Used to update a book
{
}
````
```

`ICrudAppService` has generic arguments to get the primary key type of the entity and the DTO types for the CRUD operations (it does not get the entity type since the entity type is not exposed to the clients use this interface).

> Creating an interface for an application service is good practice, but not required by the ABP Framework. You can skip the interface part.

`ICrudAppService` declares the following methods:

```
```csharp
public interface ICrudAppService<
 TEntityDto,
 in TKey,
 in TGetListInput,
 in TCreateInput,
 in TUpdateInput>
 : IApplicationService
 where TEntityDto : IEntityDto<TKey>
{
 Task<TEntityDto> GetAsync(TKey id);

 Task<PagedResultDto<TEntityDto>> GetListAsync(TGetListInput input);

 Task<TEntityDto> CreateAsync(TCreateInput input);

 Task<TEntityDto> UpdateAsync(TKey id, TUpdateInput input);

 Task DeleteAsync(TKey id);
}
````
```

DTO classes used in this example are `BookDto` and `CreateUpdateBookDto`:

```
```csharp
public class BookDto : AuditedEntityDto<Guid>
{
 public string Name { get; set; }

 public BookType Type { get; set; }

 public float Price { get; set; }
}
```

```

public class CreateUpdateBookDto
{
 [Required]
 [StringLength(128)]
 public string Name { get; set; }

 [Required]
 public BookType Type { get; set; } = BookType.Undefined;

 [Required]
 public float Price { get; set; }
}...

```

[[Profile](https://docs.automapper.org/en/stable/Configuration.html#profile-instances)](<https://docs.automapper.org/en/stable/Configuration.html#profile-instances>) class of DTO class.

```

```csharp
public class MyProfile : Profile
{
    public MyProfile()
    {
        CreateMap<Book, BookDto>();
        CreateMap<CreateUpdateBookDto, Book>();
    }
}...

```

* `CreateUpdateBookDto` is shared by create and update operations, but you could use separated DTO classes as well.

And finally, the `BookAppService` implementation is very simple:

```

```csharp
public class BookAppService :
 CrudAppService<Book, BookDto, Guid, PagedAndSortedResultRequestDto,
 CreateUpdateBookDto, CreateUpdateBookDto>,
 IBookAppService
{
 public BookAppService(IRepository<Book, Guid> repository)
 : base(repository)
 {
 }
}...

```

`CrudAppService` implements all methods declared in the `ICrudAppService` interface. You can then add your own custom methods or override and customize base methods.

> `CrudAppService` has different versions gets different number of generic arguments. Use the one suitable for you.

### ### AbstractKeyCrudAppService

`CrudAppService` requires to have an Id property as the primary key of your entity. If you are using composite keys then you can not utilize it.

`'AbstractKeyCrudAppService'` implements the same `'ICrudAppService'` interface, but this time without making assumption about your primary key.

#### #### Example

Assume that you have a `'District'` entity with `'CityId'` and `'Name'` as a composite primary key. Using `'AbstractKeyCrudAppService'` requires to implement `'DeleteByIdAsync'` and `'GetEntityByIdAsync'` methods yourself:

```
```csharp
public class DistrictAppService
    : AbstractKeyCrudAppService<District, DistrictDto, DistrictKey>
{
    public DistrictAppService(IRepository<District> repository)
        : base(repository)
    {
    }

    protected async override Task DeleteByIdAsync(DistrictKey id)
    {
        await Repository.DeleteAsync(d => d.CityId == id.CityId && d.Name == id.Name);
    }

    protected async override Task<District> GetEntityByIdAsync(DistrictKey id)
    {
        var queryable = await Repository.GetQueryableAsync();
        return await AsyncQueryableExecuter.FirstOrDefaultAsync(
            queryable.Where(d => d.CityId == id.CityId && d.Name == id.Name));
    }
}
```
...
```

This implementation requires you to create a class that represents your composite key:

```
```csharp
public class DistrictKey
{
    public Guid CityId { get; set; }

    public string Name { get; set; }
}
```
...
```

#### ### Authorization (for CRUD App Services)

There are two ways of authorizing the base application service methods;

1. You can set the policy properties (`xxxPolicyName`) in the constructor of your service. Example:

```
```csharp
public class MyPeopleAppService : CrudAppService<Person, PersonDto, Guid>
{

```

```

public MyPeopleAppService(IRepository<Person, Guid> repository)
    : base(repository)
{
    GetPolicyName = "...";
    GetListPolicyName = "...";
    CreatePolicyName = "...";
    UpdatePolicyName = "...";
    DeletePolicyName = "...";
}
...

```

`CreatePolicyName` is checked by the `CreateAsync` method and so on... You should specify a policy (permission) name defined in your application.

2. You can override the check methods (CheckXxxPolicyAsync) in your service. Example:

```

```csharp
public class MyPeopleAppService : CrudAppService<Person, PersonDto, Guid>
{
 public MyPeopleAppService(IRepository<Person, Guid> repository)
 : base(repository)
 {
 }

 protected async override Task CheckDeletePolicyAsync()
 {
 await AuthorizationService.CheckAsync("...");
 }
}
```

```

You can perform any logic in the `CheckDeletePolicyAsync` method. It is expected to throw an `AbpAuthorizationException` in any unauthorized case, like `AuthorizationService.CheckAsync` already does.

Base Properties & Methods

CRUD application service base class provides many useful base methods that **you can override** to customize it based on your requirements.

CRUD Methods

These are the essential CRUD methods. You can override any of them to completely customize the operation. Here, the definitions of the methods:

```

```csharp
Task<TGetOutputDto> GetAsync(TKey id);
Task<PagedResultDto<TGetListOutputDto>> GetListAsync(TGetListInput input);
Task<TGetOutputDto> CreateAsync(TCreateInput input);
Task<TGetOutputDto> UpdateAsync(TKey id, TUpdateInput input);
Task DeleteAsync(TKey id);
```

```

Querying

These methods are low level methods that can control how to query entities from the database.

- * `CreateFilteredQuery` can be overridden to create an `IQueryable< TEntity >` that is filtered by the given input. If your `TGetListInput` class contains any filter, it is proper to override this method and filter the query. It returns the (unfiltered) repository (which is already `IQueryable< TEntity >`) by default.
- * `ApplyPaging` is used to make paging on the query. If your `TGetListInput` already implements `IPagedResultRequest`, you don't need to override this since the ABP Framework automatically understands it and performs the paging.
- * `ApplySorting` is used to sort (order by...) the query. If your `TGetListInput` already implements the `ISortedResultRequest`, ABP Framework automatically sorts the query. If not, it fallbacks to the `ApplyDefaultSorting` which tries to sort by creation time, if your entity implements the standard `IHasCreationTime` interface.
- * `GetEntityByIdAsync` is used to get an entity by id, which calls `Repository.GetAsync(id)` by default.
- * `DeleteByIdAsync` is used to delete an entity by id, which calls `Repository.DeleteAsync(id)` by default.

Object to Object Mapping

These methods are used to convert Entities to DTOs and vice versa. They use the [IObjectMapper](Object-To-Object-Mapping.md) by default.

- * `MapToGetOutputDtoAsync` is used to map the entity to the DTO returned from the `GetAsync`, `CreateAsync` and `UpdateAsync` methods. Alternatively, you can override the `MapToGetOutputDto` if you don't need to perform any async operation.
- * `MapToGetListOutputDtosAsync` is used to map a list of entities to a list of DTOs returned from the `GetListAsync` method. It uses the `MapToGetListOutputDtoAsync` to map each entity in the list. You can override one of them based on your case. Alternatively, you can override the `MapToGetListOutputDto` if you don't need to perform any async operation.
- * `MapToEntityAsync` method has two overloads;
 - * `MapToEntityAsync(TCreateInput)` is used to create an entity from `TCreateInput`.
 - * `MapToEntityAsync(TUpdateInput, TEntity)` is used to update an existing entity from `TUpdateInput`.

Miscellaneous

Working with Streams

`Stream` object itself is not serializable. So, you may have problems if you directly use `Stream` as the parameter or the return value for your application service. ABP Framework provides a special type, `IRemoteStreamContent` to be used to get or return streams in the application services.

Example: Application Service Interface that can be used to get and return streams

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Services;
```

```

using Volo.Abp.Content;

namespace MyProject.Test
{
 public interface ITestAppService : IApplicationService
 {
 Task Upload(Guid id, IRemoteStreamContent streamContent);
 Task<IRemoteStreamContent> Download(Guid id);

 Task CreateFile(CreateFileInput input);
 Task CreateMultipleFile(CreateMultipleFileInput input);
 }

 public class CreateFileInput
 {
 public Guid Id { get; set; }

 public IRemoteStreamContent Content { get; set; }
 }

 public class CreateMultipleFileInput
 {
 public Guid Id { get; set; }

 public IEnumerable<IRemoteStreamContent> Contents { get; set; }
 }
}

***You need to configure `AbpAspNetCoreMvcOptions` to add DTO class to
`FormBodyBindingIgnoredTypes` to use `IRemoteStreamContent` in** **DTO ([Data
Transfer Object](Data-Transfer-Objects.md))**


```csharp
Configure<AbpAspNetCoreMvcOptions>(options =>
{
    options.ConventionalControllers.FormBodyBindingIgnoredTypes.Add(typeof(Create
FileInput));

    options.ConventionalControllers.FormBodyBindingIgnoredTypes.Add(typeof(Create
MultipleFileInput));
});
```

***Example: Application Service Implementation that can be used to get and
return streams***


```csharp
using System;
using System.IO;
using System.Threading.Tasks;
using Volo.Abp;
using Volo.Abp.Application.Services;
using Volo.Abp.Content;

namespace MyProject.Test
{

```

```

public class TestAppService : ApplicationService, ITestAppService
{
    public Task<IRemoteStreamContent> Download(Guid id)
    {
        var fs = new FileStream("C:\\\\Temp\\\\" + id + ".blob",
 FileMode.OpenOrCreate);
        return Task.FromResult(
            (IRemoteStreamContent) new RemoteStreamContent(fs) {
                ContentType = "application/octet-stream"
            }
        );
    }

    public async Task Upload(Guid id, IRemoteStreamContent streamContent)
    {
        using (var fs = new FileStream("C:\\\\Temp\\\\" + id + ".blob",
 FileMode.Create))
        {
            await streamContent.GetStream().CopyToAsync(fs);
            await fs.FlushAsync();
        }
    }

    public async Task CreateFileAsync(CreateFileInput input)
    {
        using (var fs = new FileStream("C:\\\\Temp\\\\" + input.Id + ".blob",
 FileMode.Create))
        {
            await input.Content.GetStream().CopyToAsync(fs);
            await fs.FlushAsync();
        }
    }

    public async Task CreateMultipleFileAsync(CreateMultipleFileInput
input)
    {
        using (var fs = new FileStream("C:\\\\Temp\\\\" + input.Id + ".blob",
 FileMode.Append))
        {
            foreach (var content in input.Contents)
            {
                await content.GetStream().CopyToAsync(fs);
            }
            await fs.FlushAsync();
        }
    }
}
...

```

`'IRemoteStreamContent'` is compatible with the [Auto API Controller](API/Auto-API-Controllers.md) and [Dynamic C# HTTP Proxy](API/Dynamic-CSharp-API-Clients.md) systems.

Lifetime

Lifetime of application services are [transient](Dependency-Injection.md) and they are automatically registered to the dependency injection system.

8.2.3.2 Data Transfer Objects

Data Transfer Objects

Introduction

Data Transfer Objects (DTO) are used to transfer data between the **Application Layer** and the **Presentation Layer** or other type of clients.

Typically, an [application service](Application-Services.md) is called from the presentation layer (optionally) with a **DTO** as the parameter. It uses domain objects to **perform some specific business logic** and (optionally) returns a DTO back to the presentation layer. Thus, the presentation layer is completely **isolated** from domain layer.

The Need for DTOs

> **You can skip this section** if you feel that you know and confirm the benefits of using DTOs.

At first, creating a DTO class for each application service method can be seen as tedious and time-consuming work. However, they can save your application if you correctly use them. Why & how?

Abstraction of the Domain Layer

DTOs provide an efficient way of **abstracting domain objects** from the presentation layer. In effect, your **layers** are correctly separated. If you want to change the presentation layer completely, you can continue with the existing application and domain layers. Alternatively, you can re-write your domain layer, completely change the database schema, entities and O/RM framework, all without changing the presentation layer. This, of course, is as long as the contracts (method signatures and DTOs) of your application services remain unchanged.

Data Hiding

Say you have a 'User' entity with the properties Id, Name, EmailAddress and Password. If a 'GetAllUsers()' method of a 'UserAppService' returns a 'List<User>', anyone can access the passwords of all your users, even if you do not show it on the screen. It's not just about security, it's about data hiding. Application services should return only what it needs by the presentation layer (or client). Not more, not less.

Serialization & Lazy Load Problems

When you return data (an object) to the presentation layer, it's most likely serialized. For example, in a REST API that returns JSON, your object will be serialized to JSON and sent to the client. Returning an Entity to the presentation layer can be problematic in that regard, especially if you are using a relational database and an ORM provider like Entity Framework Core. How?

In a real-world application, your entities may have references to each other. The 'User' entity can have a reference to its 'Role's. If you want to serialize 'User', its 'Role's are also serialized. The 'Role' class may have a 'List<Permission>' and the 'Permission' class can have a reference to a 'PermissionGroup' class and so on... Imagine all of these objects being serialized at once. You could easily and accidentally serialize your whole database! Also, if your objects have circular references, they may **not** be serialized at all.

What's the solution? Marking properties as '`NonSerialized`'? No, you can not know when it should be serialized and when it shouldn't be. It may be needed in one application service method, and not needed in another. Returning safe, serializable, and specially designed DTOs is a good choice in this situation.

Almost all O/RM frameworks support lazy-loading. It's a feature that loads entities from the database when they're needed. Say a 'User' class has a reference to a 'Role' class. When you get a 'User' from the database, the 'Role' property (or collection) is not filled. When you first read the 'Role' property, it's loaded from the database. So, if you return such an Entity to the presentation layer, it will cause it to retrieve additional entities from the database by executing additional queries. If a serialization tool reads the entity, it reads all properties recursively and again your whole database can be retrieved (if there are relations between entities).

More problems can arise if you use Entities in the presentation layer. **It's best not to reference the domain/business layer assembly in the presentation layer.**

If you are convinced about using DTOs, we can continue to what ABP Framework provides and suggests about DTOs.

> ABP doesn't force you to use DTOs, however using DTOs is **strongly suggested as a best practice**.

`## Standard Interfaces & Base Classes`

A DTO is a simple class that has no dependency and you can design it in any way. However, ABP introduces some **interfaces** to determine the **conventions** for naming **standard properties** and **base classes** to **don't repeat yourself** while declaring **common properties**.

None of them are required, but using them **simplifies and standardizes** your application code.

`### Entity Related DTOs`

You typically create DTOs corresponding to your entities, which results similar classes to your entities. ABP Framework provides some base classes to simplify while creating such DTOs.

`#### EntityDto`

'`IEntityDto<TKey>`' is a simple interface that only defines an 'Id' property. You can implement it or inherit from the '`EntityDto<TKey>`' for your DTOs that matches to an [entity](Entities.md).

Example:

```
```csharp
using System;
using Volo.Abp.Application.Dtos;

namespace AbpDemo
{
 public class ProductDto : EntityDto<Guid>
 {
 public string Name { get; set; }
 //...
 }
}
```

```

Audited DTOs

If your entity inherits from audited entity classes (or implements auditing interfaces), you can use the following base classes to create your DTOs:

- * `CreationAuditedEntityDto`
- * `CreationAuditedEntityWithUserDto`
- * `AuditedEntityDto`
- * `AuditedEntityWithUserDto`
- * `FullAuditedEntityDto`
- * `FullAuditedEntityWithUserDto`

Extensible DTOs

If you want to use the [object extension system](Object-Extensions.md) for your DTOs, you can use or inherit from the following DTO classes:

- * `ExtensibleObject` implements the `IHasExtraProperties` (other classes inherits this class).
- * `ExtensibleEntityDto`
- * `ExtensibleCreationAuditedEntityDto`
- * `ExtensibleCreationAuditedEntityWithUserDto`
- * `ExtensibleAuditedEntityDto`
- * `ExtensibleAuditedEntityWithUserDto`
- * `ExtensibleFullAuditedEntityDto`
- * `ExtensibleFullAuditedEntityWithUserDto`

List Results

It is common to return a list of DTOs to the client. `IListResult<T>` interface and `ListResultDto<T>` class is used to make it standard.

The definition of the `IListResult<T>` interface:

```
```csharp
public interface IListResult<T>
{
 IReadOnlyList<T> Items { get; set; }
}
```

```

****Example: Return a list of products****

```
```csharp

```

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace AbpDemo
{
 public class ProductAppService : ApplicationService, IProductAppService
 {
 private readonly IRepository<Product, Guid> _productRepository;

 public ProductAppService(IRepository<Product, Guid> productRepository)
 {
 _productRepository = productRepository;
 }

 public async Task<ListResultDto<ProductDto>> GetListAsync()
 {
 //Get entities from the repository
 List<Product> products = await _productRepository.GetListAsync();

 //Map entities to DTOs
 List<ProductDto> productDtos =
 ObjectMapper.Map<List<Product>, List<ProductDto>>(products);

 //Return the result
 return new ListResultDto<ProductDto>(productDtos);
 }
 }
}

```

You could simply return the `productDtos` object (and change the method return type) and it has nothing wrong. Returning a `ListResultDto` makes your `List<ProductDto>` wrapped into another object as an `Items` property. This has one advantage: You can later add more properties to your return value without breaking your remote clients (when they get the value as a JSON result). So, it is especially suggested when you are developing reusable application modules.

### ### Paged & Sorted List Results

It is more common to request a paged list from server and return a paged list to the client. ABP defines a few interface and classes to standardize it:

#### #### Input (Request) Types

The following interfaces and classes is to standardize the input sent by the clients.

- \* `ILimitedResultRequest`: Defines a `MaxResultCount` (`int`) property to request a limited result from the server.
- \* `IPagedResultRequest`: Inherits from the `ILimitedResultRequest` (so it inherently has the `MaxResultCount` property) and defines a `SkipCount`

```
(`int`) to declare the skip count while requesting a paged result from the server.
* `ISortedResultRequest`: Defines a `Sorting` (`string`) property to request a sorted result from the server. Sorting value can be "*Name*", "*Name DESC*", "*Name ASC, Age DESC*"... etc.
* `IPagedAndSortedResultRequest` inherits from both of the `IPagedResultRequest` and `ISortedResultRequest`, so has `MaxResultCount`, `SkipCount` and `Sorting` properties.
```

Instead of implementing the interfaces manually, it is suggested to inherit one of the following base DTO classes:

```
* `LimitedResultRequestDto` implements `ILimitedResultRequest`.
* `PagedResultRequestDto` implements `IPagedResultRequest` (and inherits from the `LimitedResultRequestDto`).
* `PagedAndSortedResultRequestDto` implements `IPagedAndSortedResultRequest` (and inherit from the `PagedResultRequestDto`).
```

#### ##### Max Result Count

`LimitedResultRequestDto` (and inherently the others) limits and validates the `MaxResultCount` by the following rules;

- \* If the client doesn't set `MaxResultCount`, it is assumed as \*\*10\*\* (the default page size). This value can be changed by setting the `LimitedResultRequestDto.DefaultMaxResultCount` static property.
- \* If the client sends `MaxResultCount` greater than \*\*1,000\*\*, it produces a \*\*validation error\*\*. It is important to protect the server from abuse of the service. If you want, you can change this value by setting the `LimitedResultRequestDto.MaxMaxResultCount` static property.

Static properties suggested to be set on application startup since they are static (global).

#### #### Output (Response) Types

The following interfaces and classes is to standardize the output sent to the clients.

```
* `IHasTotalCount` defines a `TotalCount` (`long`) property to return the total count of the records in case of paging.
* `IPagedResult<T>` inherits from the `ListResult<T>` and `IHasTotalCount`, so it has the `Items` and `TotalCount` properties.
```

Instead of implementing the interfaces manually, it is suggested to inherit one of the following base DTO classes:

```
* `PagedResultDto<T>` inherits from the `ListResultDto<T>` and also implements the `IPagedResult<T>`.
```

\*\*Example: Request a paged & sorted result from server and return a paged list\*\*

```
```csharp  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Linq.Dynamic.Core;
```

```

using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace AbpDemo
{
    public class ProductAppService : ApplicationService, IProductAppService
    {
        private readonly IRepository<Product, Guid> _productRepository;

        public ProductAppService(IRepository<Product, Guid>
productRepository)
        {
            _productRepository = productRepository;
        }

        public async Task<PagedResultDto<ProductDto>> GetListAsync(
            PagedAndSortedResultRequestDto input)
        {
            //Create the query
            var query = _productRepository
                .OrderBy(input.Sorting);

            //Get total count from the repository
            var totalCount = await query.CountAsync();

            //Get entities from the repository
            List<Product> products = await query
                .Skip(input.SkipCount)
                .Take(input.MaxResultCount).ToListAsync();

            //Map entities to DTOs
            List<ProductDto> productDtos =
                ObjectMapper.Map<List<Product>, List<ProductDto>>(products);

            //Return the result
            return new PagedResultDto<ProductDto>(totalCount, productDtos);
        }
    }
}
```

```

ABP Framework also defines a `PageBy` extension method (that is compatible with the `IPagedResultRequest`) that can be used instead of `Skip` + `Take` calls:

```

```csharp
var query = _productRepository
    .OrderBy(input.Sorting)
    .PageBy(input);
```

```

> Notice that we added `Volo.Abp.EntityFrameworkCore` package to the project to be able to use the `ToListAsync` and `CountAsync` methods since they are not included in the standard LINQ, but defined by the Entity Framework Core.

See also the [\[repository documentation\]](#)(Repositories.md) if you haven't understood the example code.

## ## Related Topics

### ### Validation

Inputs of [\[application service\]](#)(Application-Services.md) methods, controller actions, page model inputs... are automatically validated. You can use the standard data annotation attributes or a custom validation method to perform the validation.

See the [\[validation document\]](#)(Validation.md) for more.

### ### Object to Object Mapping

When you create a DTO that is related to an entity, you generally need to map these objects. ABP provides an object to object mapping system to simplify the mapping process. See the following documents:

- \* [\[Object to Object Mapping document\]](#)(Object-To-Object-Mapping.md) covers all the features.
- \* [\[Application Services document\]](#)(Application-Services.md) provides a full example.

## ## Best Practices

You are free to design your DTO classes. However, there are some best practices & suggestions that you may want to follow.

### ### Common Principles

- \* DTOs should be **well serializable** since they are generally serialized and deserialized (to JSON or other format). It is suggested to have an empty (parameterless) public constructor if you have another constructor with parameter(s).
- \* DTOs **should not contain any business logic**, except some formal [\[validation\]](#)(Validation.md) code.
- \* Do not inherit DTOs from entities and **do not reference to entities**. The [\[application startup template\]](#)(Startup-Templates/Application.md) already prevents it by separating the projects.
- \* If you use an auto [\[object to object mapping\]](#)(Object-To-Object-Mapping.md) library, like AutoMapper, enable the **mapping configuration validation** to prevent potential bugs.

### ### Input DTO Principles

- \* Define only the **properties needed** for the use case. Do not include properties not used for the use case, which confuses developers if you do so.
- \* **Don't reuse** input DTOs among different application service methods. Because, different use cases will need to and use different properties of the DTO which results some properties are not used in some cases and that makes harder to understand and use the services and causes potential bugs in the future.

### ### Output DTO Principles

- \* You can \*\*reuse output DTOs\*\* if you \*\*fill all the properties\*\* on all the cases.

#### 8.2.3.3 Unit Of Work

##### # Unit of Work

ABP Framework's Unit Of Work (UOW) implementation provides an abstraction and control on a \*\*database connection and transaction\*\* scope in an application.

Once a new UOW started, it creates an \*\*ambient scope\*\* that is participated by \*\*all the database operations\*\* performed in the current scope and considered as a \*\*single transaction boundary\*\*. The operations are \*\*committed\*\* (on success) or \*\*rolled back\*\* (on exception) all together.

ABP's UOW system is;

- \* \*\*Works conventional\*\*, so most of the times you don't deal with UOW at all.
- \* \*\*Database provider independent\*\*.
- \* \*\*Web independent\*\*, that means you can create unit of work scopes in any type of applications beside web applications/services.

##### ## Conventions

The following method types are considered as a unit of work:

- \* ASP.NET Core MVC \*\*Controller Actions\*\*.
- \* ASP.NET Core Razor \*\*Page Handlers\*\*.
- \* \*\*Application service\*\* methods.
- \* \*\*Repository methods\*\*.

A UOW automatically begins for these methods \*\*except\*\* if there is already a \*\*surrounding (ambient)\*\* UOW in action. Examples;

- \* If you call a [repository](Repositories.md) method and there is no UOW started yet, it automatically \*\*begins a new transactional UOW\*\* that involves all the operations done in the repository method and \*\*commits the transaction\*\* if the repository method \*\*doesn't throw any exception.\*\* The repository method doesn't know about UOW or transaction at all. It just works on a regular database objects ('`DbContext`' for [EF Core](Entity-Framework-Core.md), for example) and the UOW is handled by the ABP Framework.
- \* If you call an [application service](Application-Services.md) method, the same UOW system works just as explained above. If the application service method uses some repositories, the repositories \*\*don't begin a new UOW\*\*, but \*\*participates to the current unit of work\*\* started by the ABP Framework for the application service method.
- \* The same is true for an ASP.NET Core controller action. If the operation has started with a controller action, then the \*\*UOW scope is the controller action's method body\*\*.

All of these are automatically handled by the ABP Framework.

##### ### Database Transaction Behavior

While the section above explains the UOW as it is database transaction, actually a UOW doesn't have to be transactional. By default;

- \* \*\*HTTP GET\*\* requests don't start a transactional UOW. They still starts a UOW, but \*\*doesn't create a database transaction\*\*.
- \* All other HTTP request types start a UOW with a database transaction, if database level transactions are supported by the underlying database provider.

This is because an HTTP GET request doesn't (and shouldn't) make any change in the database. You can change this behavior using the options explained below.

#### `## Default Options`

`'AbpUnitOfWorkDefaultOptions'` is used to configure the default options for the unit of work system. Configure the options in the `'ConfigureServices'` method of your [module](Module-Development-Basics.md).

**\*\*Example: Completely disable the database transactions\*\***

```
```csharp
Configure<AbpUnitOfWorkDefaultOptions>(options =>
{
    options.TransactionBehavior = UnitOfWorkTransactionBehavior.Disabled;
});
```

`### Option Properties`

- * `'TransactionBehavior'` (`'enum': 'UnitOfWorkTransactionBehavior'`): A global point to configure the transaction behavior. Default value is `'Auto'` and work as explained in the `"*Database Transaction Behavior*` section above. You can enable (even for HTTP GET requests) or disable transactions with this option.
- * `'TimeOut'` (`'int?'`): Used to set the timeout value for UOWs. ****Default value is 'null'**** and uses to the default of the underlying database provider.
- * `'IsolationLevel'` (`'IsolationLevel?'`): Used to set the [isolation level](<https://docs.microsoft.com/en-us/dotnet/api/system.data.isolationlevel>) of the database transaction, if the UOW is transactional.

`## Controlling the Unit Of Work`

In some cases, you may want to change the conventional transaction scope, create inner scopes or fine control the transaction behavior. The following sections cover these possibilities.

`### IUnitOfWorkEnabled Interface`

This is an easy way to enable UOW for a class (or a hierarchy of classes) that is not unit of work by the conventions explained above.

****Example: Implement `'IUnitOfWorkEnabled'` for an arbitrary service****

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Uow;

namespace AbpDemo
```

```

{
 public class MyService : ITransientDependency, IUnitOfWorkEnabled
 {
 public virtual async Task FooAsync()
 {
 //this is a method with a UOW scope
 }
 }
}...

```

Then `MyService` (and any class derived from it) methods will be UOW.

However, there are \*\*some rules should be followed\*\* in order to make it working;

- \* If you are \*\*not injecting\*\* the service over an interface (like `IMyService`), then the methods of the service must be `virtual` (otherwise, [dynamic proxy / interception](Dynamic-Proxying-Interceptors.md) system can not work).
  - \* Only `async` methods (methods returning a `Task` or `Task<T>`) are intercepted. So, sync methods can not start a UOW.
- > Notice that if `FooAsync` is called inside a UOW scope, then it already participates to the UOW without needing to the `IUnitOfWorkEnabled` or any other configuration.

### ### UnitOfWorkAttribute

`UnitOfWork` attribute provides much more possibility like enabling or disabling UOW and controlling the transaction behavior.

`UnitOfWork` attribute can be used for a \*\*class\*\* or a \*\*method\*\* level.

**Example:** Enable UOW for a specific method of a class\*\*

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Uow;

namespace AbpDemo
{
    public class MyService : ITransientDependency
    {
        [UnitOfWork]
        public virtual async Task FooAsync()
        {
            //this is a method with a UOW scope
        }

        public virtual async Task BarAsync()
        {
            //this is a method without UOW
        }
    }
}...

```

Example: Enable UOW for all the methods of a class

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Uow;

namespace AbpDemo
{
 [UnitOfWork]
 public class MyService : ITransientDependency
 {
 public virtual async Task FooAsync()
 {
 //this is a method with a UOW scope
 }

 public virtual async Task BarAsync()
 {
 //this is a method with a UOW scope
 }
 }
}
```

```

Again, the **same rules** are valid here:

- * If you are **not injecting** the service over an interface (like `IMyService`), then the methods of the service must be `virtual` (otherwise, [dynamic proxy / interception](Dynamic-Proxying-Interceptors.md) system can not work).
- * Only `async` methods (methods returning a `Task` or `Task<T>`) are intercepted. So, sync methods can not start a UOW.

UnitOfWorkAttribute Properties

- * `IsTransactional` (`bool?`): Used to set whether the UOW should be transactional or not. **Default value is `null`**. if you leave it `null`, it is determined automatically based on the conventions and the configuration.
- * `TimeOut` (`int?`): Used to set the timeout value for this UOW. **Default value is `null`** and fallbacks to the default configured value.
- * `IsolationLevel` (`IsolationLevel?`): Used to set the [isolation level](https://docs.microsoft.com/en-us/dotnet/api/system.data.isolationlevel) of the database transaction, if the UOW is transactional. If not set, uses the default configured value.
- * `IsDisabled` (`bool`): Used to disable the UOW for the current method/class.

> If a method is called in an ambient UOW scope, then the `UnitOfWork` attribute is ignored and the method participates to the surrounding transaction in any way.

Example: Disable UOW for a controller action

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.AspNetCore.Mvc;
```

```

using Volo.Abp.Uow;

namespace AbpDemo.Web
{
 public class MyController : AbpController
 {
 [UnitOfWork(IsDisabled = true)]
 public virtual async Task FooAsync()
 {
 //...
 }
 }
}

IUnitOfWorkManager

`IUnitOfWorkManager` is the main service that is used to control the unit of work system. The following sections explains how to directly work with this service (while most of the times you won't need).

Begin a New Unit Of Work

`IUnitOfWorkManager.Begin` method is used to create a new UOW scope.

Example: Create a new non-transactional UOW scope

```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Uow;

namespace AbpDemo
{
    public class MyService : ITransientDependency
    {
        private readonly IUnitOfWorkManager _unitOfWorkManager;

        public MyService(IUnitOfWorkManager unitOfWorkManager)
        {
            _unitOfWorkManager = unitOfWorkManager;
        }

        public virtual async Task FooAsync()
        {
            using (var uow = _unitOfWorkManager.Begin(
                requiresNew: true, isTransactional: false
            ))
            {
                //...

                await uow.CompleteAsync();
            }
        }
    }
}
```

```

`'Begin'` method gets the following optional parameters:

- \* `'requiresNew' ('bool')`: Set `'true'` to ignore the surrounding unit of work and start a new UOW with the provided options. \*\*Default value is `'false'`. If it is `'false'` and there is a surrounding UOW, `'Begin'` method doesn't actually begin a new UOW, but silently participates to the existing UOW.\*\*
- \* `'isTransactional' ('bool')`: Default value is `'false'`.
- \* `'isolationLevel' ('IsolationLevel?')`: Used to set the [isolation level](<https://docs.microsoft.com/en-us/dotnet/api/system.data.isolationlevel>) of the database transaction, if the UOW is transactional. If not set, uses the default configured value.
- \* `'TimeOut' ('int?')`: Used to set the timeout value for this UOW. \*\*Default value is `'null'`\*\* and fallbacks to the default configured value.

#### #### The Current Unit Of Work

UOW is ambient, as explained before. If you need to access to the current unit of work, you can use the `'IUnitOfWorkManager.Current'` property.

\*\*Example: Get the current UOW\*\*

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Uow;

namespace AbpDemo
{
    public class MyProductService : ITransientDependency
    {
        private readonly IUnitOfWorkManager _unitOfWorkManager;

        public MyProductService(IUnitOfWorkManager unitOfWorkManager)
        {
            _unitOfWorkManager = unitOfWorkManager;
        }

        public async Task FooAsync()
        {
            var uow = _unitOfWorkManager.Current;
            //...
        }
    }
}...```

```

`'Current'` property returns a `'IUnitOfWork'` object.

> **Current Unit Of Work can be `'null'`** if there is no surrounding unit of work. It won't be `'null'` if your class is a conventional UOW class, you manually made it UOW or it was called inside a UOW scope, as explained before.

SaveChangesAsync

`'IUnitOfWork.SaveChangesAsync()'` method can be needed to save all the changes until now to the database. If you are using EF Core, it behaves exactly same.

If the current UOW is transactional, even saved changes can be rolled back on an error (for the supporting database providers).

Example: Save changes after inserting an entity to get its auto-increment id

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.Application.Services;
using Volo.Abp.Domain.Repositories;

namespace AbpDemo
{
 public class CategoryAppService : ApplicationService, ICategoryAppService
 {
 private readonly IRepository<Category, int> _categoryRepository;

 public CategoryAppService(IRepository<Category, int> categoryRepository)
 {
 _categoryRepository = categoryRepository;
 }

 public async Task<int> CreateAsync(string name)
 {
 var category = new Category {Name = name};
 await _categoryRepository.InsertAsync(category);

 //Saving changes to be able to get the auto increment id
 await UnitOfWorkManager.Current.SaveChangesAsync();

 return category.Id;
 }
 }
}...```

```

This example uses auto-increment `int` primary key for the `Category` [entity](Entities.md). Auto-increment PKs require to save the entity to the database to get the id of the new entity.

This example is an [application service](Application-Services.md) derived from the base `ApplicationService` class, which already has the `IUnitOfWorkManager` service injected as the `UnitOfWorkManager` property. So, no need to inject it manually.

Since getting the current UOW is pretty common, there is also a `CurrentUnitOfWork` property as a shortcut to the `UnitOfWorkManager.Current`. So, the example above can be changed to use it:

```
```csharp
await CurrentUnitOfWork.SaveChangesAsync();
```

```

#### ##### Alternative to the SaveChanges()

Since saving changes after inserting, updating or deleting an entity can be frequently needed, corresponding [repository](Repositories.md) methods has an

optional `autoSave` parameter. So, the `CreateAsync` method above could be re-written as shown below:

```
```csharp
public async Task<int> CreateAsync(string name)
{
    var category = new Category {Name = name};
    await _categoryRepository.InsertAsync(category, autoSave: true);
    return category.Id;
}...```

```

If your intent is just to save the changes after creating/updating/deleting an entity, it is suggested to use the `autoSave` option instead of manually using the `CurrentUnitOfWork.SaveChangesAsync()`.

- > **Note-1**: All changes are automatically saved when a unit of work ends without any error. So, don't call `SaveChangesAsync()` and don't set `autoSave` to `true` unless you really need it.
- >
- > **Note-2**: If you use `Guid` as the primary key, you never need to save changes on insert to just get the generated id, because `Guid` keys are set in the application and are immediately available once you create a new entity.

Other IUnitOfWork Properties/Methods

- * `OnCompleted` method gets a callback action which is called when the unit of work successfully completed (where you can be sure that all changes are saved).
- * `Failed` and `Disposed` events can be used to be notified if the UOW fails or when it is disposed.
- * `Complete` and `Rollback` methods are used to complete (commit) or roll backs the current UOW, which are normally used internally by the ABP Framework but can be used if you manually start a transaction using the `IUnitOfWorkManager.Begin` method.
- * `Options` can be used to get options that was used while starting the UOW.
- * `Items` dictionary can be used to store and get arbitrary objects inside the same unit of work, which can be a point to implement custom logics.

ASP.NET Core Integration

Unit of work system is fully integrated to the ASP.NET Core. It properly works when you use ASP.NET Core MVC Controllers or Razor Pages. It defines action filters and page filters for the UOW system.

- > You typically do nothing to configure the UOW when you use ASP.NET Core.

Unit Of Work Middleware

`AbpUnitOfWorkMiddleware` is a middleware that can enable UOW in the ASP.NET Core request pipeline. This might be needed if you need to enlarge the UOW scope to cover some other middleware(s).

Example:

```
```csharp
app.UseUnitOfWork();```

```

```
app.UseConfiguredEndpoints();
```

## 8.2.4 E-Book: Implementing DDD

"<https://abp.io/books/implementing-domain-driven-design>"

## 8.3 Multi Tenancy

### # Multi-Tenancy

Multi-Tenancy is a widely used architecture to create \*\*SaaS applications\*\* where the hardware and software \*\*resources are shared by the customers\*\* (tenants). ABP Framework provides all the base functionalities to create \*\*multi tenant applications\*\*.

Wikipedia [defines](<https://en.wikipedia.org/wiki/Multitenancy>) the multi-tenancy as like that:

> Software \*\*Multi-tenancy\*\* refers to a software \*\*architecture\*\* in which a \*\*single instance\*\* of software runs on a server and serves \*\*multiple tenants\*\*. A tenant is a group of users who share a common access with specific privileges to the software instance. With a multitenant architecture, a software application is designed to provide every tenant a \*\*dedicated share of the instance including its data\*\*, configuration, user management, tenant individual functionality and non-functional properties. Multi-tenancy contrasts with multi-instance architectures, where separate software instances operate on behalf of different tenants.

### ## Terminology: Host vs Tenant

There are two main side of a typical SaaS / Multi-tenant application:

- \* A \*\*Tenant\*\* is a customer of the SaaS application that pays money to use the service.
- \* \*\*Host\*\* is the company that owns the SaaS application and manages the system.

The Host and the Tenant terms will be used for that purpose in the rest of the document.

### ## Configuration

#### ### AbpMultiTenancyOptions: Enable/Disable Multi-Tenancy

'`AbpMultiTenancyOptions`' is the main options class to \*\*enable/disable the multi-tenancy\*\* for your application.

\*\*Example: Enable multi-tenancy\*\*

```
```csharp
Configure<AbpMultiTenancyOptions>(options =>
{
```

```
        options.IsEnabled = true;
});
```

> Multi-Tenancy is disabled in the ABP Framework by default. However, it is **enabled by default** when you create a new solution using the [startup template](Startup-Templates/Application.md). `MultiTenancyConsts` class in the solution has a constant to control it in a single place.

Database Architecture

ABP Framework supports all the following approaches to store the tenant data in the database;

- * **Single Database**: All tenants are stored in a single database.
- * **Database per Tenant**: Every tenant has a separate, dedicated database to store the data related to that tenant.
- * **Hybrid**: Some tenants share a single databases while some tenants may have their own databases.

[Tenant management module](Modules/Tenant-Management.md) (which comes pre-installed with the startup projects) allows you to set a connection string for any tenant (as optional), so you can achieve any of the approaches.

Usage

Multi-tenancy system is designed to **work seamlessly** and make your application code **multi-tenancy unaware** as much as possible.

IMultiTenant

You should implement the `IMultiTenant` interface for your [entities](Entities.md) to make them **multi-tenancy ready**.

Example: A multi-tenant *Product* entity

```
```csharp
using System;
using Volo.Abp.Domain.Entities;
using Volo.Abp.MultiTenancy;

namespace MultiTenancyDemo.Products
{
 public class Product : AggregateRoot<Guid>, IMultiTenant
 {
 public Guid? TenantId { get; set; } //Defined by the IMultiTenant
interface

 public string Name { get; set; }

 public float Price { get; set; }
 }
}...```

```

`IMultiTenant` interface just defines a `TenantId` property. When you implement this interface, ABP Framework \*\*automatically\*\* [filters](Data-Filtering.md) entities for the current tenant when you query from database.

So, you don't need to manually add '`TenantId`' condition while performing queries. A tenant can not access to data of another tenant by default.

#### #### Why the `TenantId` Property is Nullable?

`IMultiTenant.TenantId` is **nullable**. When it is null that means the entity is owned by the **Host** side and not owned by a tenant. It is useful when you create a functionality in your system that is both used by the tenant and the host sides.

For example, '`IdentityUser`' is an entity defined by the [\[Identity Module\]](#)(Modules/Identity.md). The host and all the tenants have their own users. So, for the host side, users will have a '`null`' '`TenantId`' while tenant users will have their related '`TenantId`'.

> **Tip:** If your entity is tenant-specific and has no meaning in the host side, you can force to not set '`null`' for the '`TenantId`' in the constructor of your entity.

#### #### When to set the `TenantId`?

ABP automatically sets the '`TenantId`' for you when you create a new entity object. It is done in the constructor of the base '`Entity`' class (all other base entity and aggregate root classes are derived from the '`Entity`' class). The '`TenantId`' is set from the current value of the '`ICurrentTenant.Id`' property (see the next section).

If you set the '`TenantId`' value for a specific entity object, it will override the value set by the base class. If you want to set the '`TenantId`' property yourself, we recommend to do it in the constructor of your entity class and do not change (update) it again (Actually, changing it means that you are moving the entity from a tenant to another tenant. If you want that, you need an extra care about the related entities in the database).

#### ### `ICurrentTenant`

`ICurrentTenant` is the main service to interact with the multi-tenancy infrastructure.

`ApplicationService`, `DomainService`, `AbpController` and some other base classes already has pre-injected '`CurrentTenant`' properties. For other type of classes, you can inject the '`ICurrentTenant`' into your service.

#### #### Tenant Properties

`ICurrentTenant` defines the following properties;

- \* `Id` (`Guid`): Id of the current tenant. Can be '`null`' if the current user is a host user or the tenant could not be determined from the request.
- \* `Name` (`string`): Name of the current tenant. Can be '`null`' if the current user is a host user or the tenant could not be determined from the request.
- \* `IsAvailable` (`bool`): Returns '`true`' if the '`Id`' is not '`null`'.

#### #### Change the Current Tenant

ABP Framework automatically filters the resources (database, cache...) based on the '`ICurrentTenant.Id`'. However, in some cases you may want to perform an

operation on behalf of a specific tenant, generally when you are in the host context.

``ICurrentTenant.Change`` method changes the current tenant for a limited scope, so you can safely perform operations for the tenant.

\*\*Example: Get product count of a specific tenant\*\*

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Domain.Services;

namespace MultiTenancyDemo.Products
{
    public class ProductManager : DomainService
    {
        private readonly IRepository<Product, Guid> _productRepository;

        public ProductManager(IRepository<Product, Guid> productRepository)
        {
            _productRepository = productRepository;
        }

        public async Task<long> GetProductCountAsync(Guid? tenantId)
        {
            using (CurrentTenant.Change(tenantId))
            {
                return await _productRepository.GetCountAsync();
            }
        }
    }
}...```

```

- * `Change` method can be used in a **nested way**. It restores the `CurrentTenant.Id` to the previous value after the `using` statement.
- * When you use `CurrentTenant.Id` inside the `Change` scope, you get the `tenantId` provided to the `Change` method. So, the repository also get this `tenantId` and can filter the database query accordingly.
- * Use `CurrentTenant.Change(null)` to change scope to the host context.

> Always use the `Change` method with a `using` statement like done in this example.

Data Filtering: Disable the Multi-Tenancy Filter

As mentioned before, ABP Framework handles data isolation between tenants using the [Data Filtering](Data-Filtering.md) system. In some cases, you may want to disable it and perform a query on all the data, without filtering for the current tenant.

Example: Get count of products in the database, including all the products of all the tenants.

```
```csharp
using System;```

```

```

using System.Threading.Tasks;
using Volo.Abp.Data;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Domain.Services;
using Volo.Abp.MultiTenancy;

namespace MultiTenancyDemo.Products
{
 public class ProductManager : DomainService
 {
 private readonly IRepository<Product, Guid> _productRepository;
 private readonly IDataFilter _dataFilter;

 public ProductManager(
 IRepository<Product, Guid> productRepository,
 IDataFilter dataFilter)
 {
 _productRepository = productRepository;
 _dataFilter = dataFilter;
 }

 public async Task<long> GetProductCountAsync()
 {
 using (_dataFilter.Disable<IMultiTenant>())
 {
 return await _productRepository.GetCountAsync();
 }
 }
 }
}
...

```

See the [\[Data Filtering document\]](#)(Data-Filtering.md) for more.

> Note that this approach won't work if your tenants have \*\*separate databases\*\* since there is no built-in way to query from multiple database in a single database query. You should handle it yourself if you need it.

## ## Infrastructure

### ### Determining the Current Tenant

The first thing for a multi-tenant application is to determine the current tenant on the runtime.

ABP Framework provides an extensible \*\*Tenant Resolving\*\* system for that purpose. Tenant Resolving system then used in the \*\*Multi-Tenancy Middleware\*\* to determine the current tenant for the current HTTP request.

#### #### Tenant Resolvers

#### ##### Default Tenant Resolvers

The following resolvers are provided and configured by default;

```
* `CurrentUserTenantResolveContributor`: Gets the tenant id from claims of the current user, if the current user has logged in. **This should always be the first contributor for the security**.
* `QueryStringTenantResolveContributor`: Tries to find current tenant id from query string parameters. The parameter name is `__tenant` by default.
* `RouteTenantResolveContributor`: Tries to find current tenant id from route (URL path). The variable name is `__tenant` by default. If you defined a route with this variable, then it can determine the current tenant from the route.
* `HeaderTenantResolveContributor`: Tries to find current tenant id from HTTP headers. The header name is `__tenant` by default.
* `CookieTenantResolveContributor`: Tries to find current tenant id from cookie values. The cookie name is `__tenant` by default.
```

#### ##### Problems with the NGINX

You may have problems with the `\_\_tenant` in the HTTP Headers if you're using the [nginx](https://www.nginx.com/) as the reverse proxy server. Because it doesn't allow to use underscore and some other special characters in the HTTP headers and you may need to manually configure it. See the following documents please:

[http://nginx.org/en/docs/http/ngx\\_http\\_core\\_module.html#ignore\\_invalid\\_headers](http://nginx.org/en/docs/http/ngx_http_core_module.html#ignore_invalid_headers)  
[http://nginx.org/en/docs/http/ngx\\_http\\_core\\_module.html#underscores\\_in\\_headers](http://nginx.org/en/docs/http/ngx_http_core_module.html#underscores_in_headers)

#### ##### AbpAspNetCoreMultiTenancyOptions

`\_\_tenant` parameter name can be changed using `AbpAspNetCoreMultiTenancyOptions`.

\*\*Example:\*\*

```
```csharp
services.Configure<AbpAspNetCoreMultiTenancyOptions>(options =>
{
    options.TenantKey = "MyTenantKey";
});
```

If you change the `TenantKey`, make sure to pass it to `CoreModule` in the Angular client as follows:

```
```js
@NgModule({
 imports: [
 CoreModule.forRoot({
 // ...
 tenantKey: 'MyTenantKey'
 }),
 // ...
 }
)
export class AppModule {}
```

If you need to access it, you can inject it as follows:

```

```js
import { Inject } from '@angular/core';
import { TENANT_KEY } from '@abp/ng.core';

class SomeComponent {
    constructor(@Inject(TENANT_KEY) private tenantKey: string) {}
}..
```
> However, we don't suggest to change this value since some clients may assume the the `__tenant` as the parameter name and they might need to manually configure then.

```

The `MultiTenancyMiddlewareErrorPageBuilder` is used to handle inactive and non-existent tenants.

It will respond to an error page by default, you can change it if you want, eg: only output the error log and continue ASP.NET Core's request pipeline.

```

```csharp
Configure<AbpAspNetCoreMultiTenancyOptions>(options =>
{
    options.MultiTenancyMiddlewareErrorPageBuilder = async (context,
exception) =>
    {
        // Handle the exception.

        // Return true to stop the pipeline, false to continue.
        return true;
    };
}..
```

```

#### ##### Domain/Subdomain Tenant Resolver

In a real application, most of times you will want to determine current tenant either by subdomain (like mytenant1.mydomain.com) or by the whole domain (like mytenant.com). If so, you can configure the `AbpTenantResolveOptions` to add the domain tenant resolver.

**\*\*Example: Add a subdomain resolver\*\***

```

```csharp
Configure<AbpTenantResolveOptions>(options =>
{
    options.AddDomainTenantResolver("{0}.mydomain.com");
}..
```

```

- \* `{0}` is the placeholder to determine current tenant's unique name.
- \* Add this code to the `ConfigureServices` method of your [module](Module-Development-Basics.md).
- \* This should be done in the \*Web/API Layer\* since the URL is a web related stuff.

> There is an [example](https://github.com/abpframework/abp-samples/tree/master/DomainTenantResolver) that uses the subdomain to determining the current tenant.

#### ##### Custom Tenant Resolvers

You can add implement your custom tenant resolver and configure the `AbpTenantResolveOptions` in your module's `ConfigureServices` method as like below:

```
```csharp
Configure<AbpTenantResolveOptions>(options =>
{
    options.TenantResolvers.Add(new MyCustomTenantResolveContributor());
});```

```

`MyCustomTenantResolveContributor` must inherit from the `TenantResolveContributorBase` (or implement the `ITenantResolveContributor`) as shown below:

```
```csharp
using System.Threading.Tasks;
using Volo.Abp.MultiTenancy;

namespace MultiTenancyDemo.Web
{
 public class MyCustomTenantResolveContributor : TenantResolveContributorBase
 {
 public override string Name => "Custom";

 public override Task ResolveAsync(ITenantResolveContext context)
 {
 //TODO...
 }
 }
}...```

```

\* A tenant resolver should set `context.TenantIdOrName` if it can determine it. If not, just leave it as is to allow the next resolver to determine it.  
\* `context.ServiceProvider` can be used if you need to additional services to resolve from the [dependency injection](Dependency-Injection.md) system.

#### #### Multi-Tenancy Middleware

Multi-Tenancy middleware is an ASP.NET Core request pipeline [middleware](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware) that determines the current tenant from the HTTP request and sets the `ICurrentTenant` properties.

Multi-Tenancy middleware is typically placed just under the [authentication](https://docs.microsoft.com/en-us/aspnet/core/security/authentication) middleware (`app.UseAuthentication()`):

```
```csharp
app.UseMultiTenancy();```

```

> This middleware is already configured in the startup templates, so you normally don't need to manually add it.

Tenant Store

'`ITenantStore`' is used to get the tenant configuration from a data source.

Tenant Management Module

The [`tenant management module`](Modules/Tenant-Management) is **included in the startup templates** and implements the '`ITenantStore`' interface to get the tenants and their configuration from a database. It also provides the necessary functionality and UI to manage the tenants and their connection strings.

Configuration Data Store

If you don't want to use the tenant management module, the '`DefaultTenantStore`' is used as the '`ITenantStore`' implementation. It gets the tenant configurations from the [`configuration system`](Configuration.md) ('`IConfiguration`'). You can either configure the '`AbpDefaultTenantStoreOptions`' [`options`](Options.md) or set it in your '`appsettings.json`' file:

Example: Define tenants in `appsettings.json`

```
```json
"Tenants": [
 {
 "Id": "446a5211-3d72-4339-9adc-845151f8ada0",
 "Name": "tenant1"
 },
 {
 "Id": "25388015-ef1c-4355-9c18-f6b6ddbaf89d",
 "Name": "tenant2",
 "ConnectionStrings": {
 "Default": "...tenant2's db connection string here..."
 }
 }
]
...```

```

> It is recommended to \*\*use the Tenant Management module\*\*, which is already pre-configured when you create a new application with the ABP startup templates.

### ### Other Multi-Tenancy Infrastructure

ABP Framework was designed to respect to the multi-tenancy in every aspect and most of the times everything will work as expected.

BLOB Storing, Caching, Data Filtering, Data Seeding, Authorization and all the other services are designed to properly work in a multi-tenant system.

## ## The Tenant Management Module

ABP Framework provides all the the infrastructure to create a multi-tenant application, but doesn't make any assumption about how you manage (create, delete...) your tenants.

The [Tenant Management module](Modules/Tenant-Management.md) provides a basic UI to manage your tenants and set their connection strings. It is pre-configured for the [application startup template](Startup-Templates/Application.md).

#### ## See Also

- \* [Features](Features.md)

## 8.4 Microservices

### # Microservice Architecture

*"Microservices are a software development technique—a variant of the service-oriented architecture (SOA) architectural style that structures an application as a collection of loosely coupled services. In a microservices architecture, services are fine-grained and the protocols are lightweight. The benefit of decomposing an application into different smaller services is that it improves modularity. This makes the application easier to understand, develop, test, and become more resilient to architecture erosion. It parallelizes development by enabling small autonomous teams to develop, deploy and scale their respective services independently. It also allows the architecture of an individual service to emerge through continuous refactoring. Microservices-based architectures enable continuous delivery and deployment."\**

- [Wikipedia](<https://en.wikipedia.org/wiki/Microservices>)

#### ## Introduction

One of the major goals of the ABP framework is to provide a convenient infrastructure to create microservice solutions. To make this possible,

- \* Provides a [module system](Module-Development-Basics.md) that allows you to split your application into modules where each module may have its own database, entities, services, APIs, UI components/pages... etc.
- \* Offers an [architectural model](Best-Practices/Module-Architecture.md) to develop your modules to be compatible to microservice development and deployment.
- \* Provides [best practices guide](Best-Practices/Index.md) to develop your module standards-compliance.
- \* Provides base infrastructure to implement [Domain Driven Design](Domain-Driven-Design.md) in your microservices.
- \* Provide services to [automatically create REST-style APIs](API/Auto-API-Controllers.md) from your application services.
- \* Provide services to [automatically create C# API clients](API/Dynamic-CSharp-API-Clients.md) that makes easy to consume your services from another service/application.
- \* Provides a [distributed event bus](Event-Bus.md) to communicate your services.
- \* Provides many other services to make your daily development easier.

## ## Microservice for New Applications

One common advise to start a new solution is \*\*always to start with a monolith\*\*, keep it modular and split into microservices once the monolith becomes a problem. This makes your progress fast in the beginning especially if your team is small and you don't want to deal with challenges of the microservice architecture.

However, developing such a well-modular application can be a problem since it is \*\*hard to keep modules isolated\*\* from each other as you would do it for microservices (see [[Stefan Tilkov's article](#)](<https://martinfowler.com/articles/dont-start-monolith.html>) about that). Microservice architecture naturally forces you to develop well isolated services, but in a modular monolithic application it's easy to tightly couple modules to each other and design \*\*weak module boundaries\*\* and API contracts.

ABP can help you in that point by offering a \*\*microservice-compatible, strict module architecture\*\* where your module is split into multiple layers/projects and developed in its own VS solution completely isolated and independent from other modules. Such a developed module is a natural microservice yet it can be easily plugged-in a monolithic application. See the [[module development best practice guide](#)](Best-Practices/Index.md) that offers a \*\*microservice-first module design\*\*. All [[standard ABP modules](#)](<https://github.com/abpframework/abp/tree/master/modules>) are developed based on this guide. So, you can use these modules by embedding into your monolithic solution or deploy them separately and use via remote APIs. They can share a single database or can have their own database based on your simple configuration.

## ## Microservice Demo Solution: eShopOnAbp

The [[eShopOnAbp project](#)](<https://github.com/abpframework/eShopOnAbp>) demonstrates a complete microservice solution based on the ABP framework.

# 9 API

## 9.1 ABP Endpoints

### 9.1.1 Application Configuration

#### # Application Configuration Endpoint

ABP Framework provides a pre-built and standard endpoint that contains some useful information about the application/service. Here, the list of some fundamental information at this endpoint:

- \* Granted [[policies](#)]([../Authorization.md](#)) (permissions) for the current user.
- \* [[Setting](#)]([../Settings.md](#)) values for the current user.
- \* Info about the [[current user](#)]([../CurrentUser.md](#)) (like id and user name).
- \* Info about the current [[tenant](#)]([../Multi-Tenancy.md](#)) (like id and name).
- \* [[Time zone](#)]([../Timing.md](#)) information for the current user and the [[clock](#)]([../Timing.md](#)) type of the application.

> If you have started with ABP's startup solution templates and using one of the official UI options, then all these are set up for you and you don't need

to know these details. However, if you are building a UI application from scratch, you may want to know this endpoint.

#### ## HTTP API

If you navigate to the `/api/abp/application-configuration` URL of an ABP Framework based web application or HTTP Service, you can access the configuration as a JSON object. This endpoint is useful to create the client of your application.

#### ## Script

For ASP.NET Core MVC (Razor Pages) applications, the same configuration values are also available on the JavaScript side.

`/Abp/ApplicationConfigurationScript` is the URL of the script that is auto-generated based on the HTTP API above.

See the [\[JavaScript API document\]](#)(../UI/AspNetCore/JavaScript-API/Index.md) for the ASP.NET Core UI.

Other UI types provide services native to the related platform. For example, see the [\[Angular UI settings documentation\]](#)(../UI/Angular/Settings.md) to learn how to use the setting values exposes by this endpoint.

### 9.1.2 Application Localization

#### # Application Localization Endpoint

ABP Framework provides a pre-built and standard endpoint that returns all the [\[localization\]](#)(../Localization.md) resources and texts defined in the server.

> If you have started with ABP's startup solution templates and using one of the official UI options, then all these are set up for you and you don't need to know these details. However, if you are building a UI application from scratch, you may want to know this endpoint.

#### ## HTTP API

`/api/abp/application-localization` is the main URL of the HTTP API that returns the localization data as a JSON string. It accepts the following query string parameters:

- \* `cultureName` (required): A culture code to get the localization data, like `en` or `en-US`.
- \* `onlyDynamics` (optional, default: `false`): Can be set to `true` to only get the dynamically defined localization resources and texts. If your client-side application shares the same localization resources with the server (like ABP's Blazor and MVC UIs), you can set `onlyDynamics` to `true` .

\*\*Example request:\*\*

....

/api/abp/application-localization?cultureName=en

```

Script

For [ASP.NET Core MVC (Razor Pages)](../UI/AspNetCore/Overall.md)
applications, the same localization data is also available on the JavaScript
side. `/Abp/ApplicationLocalizationScript` is the URL of the script that is
auto-generated based on the HTTP API above.

Example request:

```
/Abp/ApplicationLocalizationScript?cultureName=en
```

```

See the [JavaScript API document](../UI/AspNetCore/JavaScript-API/Index.md) for the ASP.NET Core UI.

Other UI types provide services native to the related platform. For example, see the [Angular UI localization documentation](../UI/Angular/Localization.md) to learn how to use the localization values exposes by this endpoint.

## 9.2 API Versioning

```

API Versioning System

ABP Framework integrates the [ASPNET-API-Versioning](https://github.com/dotnet/aspnet-api-versioning/wiki) feature and
adapts to C# and JavaScript Static Client Proxies and [Auto API
Controller](API/Auto-API-Controllers.md).

```

```

Enable API Versioning

```cs
public override void ConfigureServices(ServiceConfigurationContext context)
{
    context.Services.AddAbpApiVersioning(options =>
    {
        // Show neutral/versionless APIs.
        options.UseApiBehavior = false;

        options.ReportApiVersions = true;
        options.AssumeDefaultVersionWhenUnspecified = true;
    });

    Configure<AbpAspNetCoreMvcOptions>(options =>
    {
        options.ChangeControllerModelApiExplorerGroupName = false;
    });
}

## C# and JavaScript Static Client Proxies

```

This feature does not compatible with [URL Path Versioning](<https://github.com/dotnet/aspnet-api-versioning/wiki/Versioning-via-the-URL-Path>), we suggest to use [Versioning-via-the-Query-String](<https://github.com/dotnet/aspnet-api-versioning/wiki/Versioning-via-the-Query-String>).

Example

```
**Application Services:**  
```cs  
public interface IBookAppService : IApplicationService
{
 Task<BookDto> GetAsync();
}

public interface IBookV2AppService : IApplicationService
{
 Task<BookDto> GetAsync();

 Task<BookDto> GetAsync(string isbn);
}
...

HttpApi Controllers:
```cs  
[Area(BookStoreRemoteServiceConsts.ModuleName)]  
[RemoteService(Name = BookStoreRemoteServiceConsts.RemoteServiceName)]  
[ApiVersion("1.0", Deprecated = true)]  
[ApiController]  
[ControllerName("Book")]  
[Route("api/BookStore/Book")]  
public class BookController : BookStoreController, IBookAppService  
{  
    private readonly IBookAppService _bookAppService;  
  
    public BookController(IBookAppService bookAppService)  
    {  
        _bookAppService = bookAppService;  
    }  
  
    [HttpGet]  
    public async Task<BookDto> GetAsync()  
    {  
        return await _bookAppService.GetAsync();  
    }  
}  
  
[Area(BookStoreRemoteServiceConsts.ModuleName)]  
[RemoteService(Name = BookStoreRemoteServiceConsts.RemoteServiceName)]  
[ApiVersion("2.0")]  
[ApiController]  
[ControllerName("Book")]  
[Route("api/BookStore/Book")]  
public class BookV2Controller : BookStoreController, IBookV2AppService  
{  
    private readonly IBookV2AppService _bookAppService;  
  
    public BookV2Controller(IBookV2AppService bookAppService)
```

```

{
    _bookAppService = bookAppService;
}

[HttpGet]
public async Task<BookDto> GetAsync()
{
    return await _bookAppService.GetAsync();
}

[HttpGet]
[Route("{isbn}")]
public async Task<BookDto> GetAsync(string isbn)
{
    return await _bookAppService.GetAsync(isbn);
}
...
```

```

**\*\*Generated CS and JS proxies:\*\***

```

```cs
[Dependency(ReplaceServices = true)]
[ExposeServices(typeof(IBookAppService), typeof(BookClientProxy))]
public partial class BookClientProxy : ClientProxyBase<IBookAppService>, IBookAppService
{
    public virtual async Task<BookDto> GetAsync()
    {
        return await RequestAsync<BookDto>(nameof(GetAsync));
    }
}

[Dependency(ReplaceServices = true)]
[ExposeServices(typeof(IBookV2AppService), typeof(BookV2ClientProxy))]
public partial class BookV2ClientProxy : ClientProxyBase<IBookV2AppService>, IBookV2AppService
{
    public virtual async Task<BookDto> GetAsync()
    {
        return await RequestAsync<BookDto>(nameof(GetAsync));
    }

    public virtual async Task<BookDto> GetAsync(string isbn)
    {
        return await RequestAsync<BookDto>(nameof(GetAsync), new ClientProxyRequestTypeValue
        {
            { typeof(string), isbn }
        });
    }
}
```

```js
// controller bookStore.books.book
```

```

```

(function(){

abp.utils.createNamespace(window, 'bookStore.books.book');

bookStore.books.book.get = function(api_version, ajaxParams) {
 var api_version = api_version ? api_version : '1.0';
 return abp.ajax($.extend(true, {
 url: abp.appPath + 'api/BookStore/Book' +
 abp.utils.buildQueryString([{ name: 'api-version', value: api_version }]) +
 '',
 type: 'GET'
 }, ajaxParams));
};

})();

// controller bookStore.books.bookV2

(function(){

abp.utils.createNamespace(window, 'bookStore.books.bookV2');

bookStore.books.bookV2.get = function(api_version, ajaxParams) {
 var api_version = api_version ? api_version : '2.0';
 return abp.ajax($.extend(true, {
 url: abp.appPath + 'api/BookStore/Book' +
 abp.utils.buildQueryString([{ name: 'api-version', value: api_version }]) +
 '',
 type: 'GET'
 }, ajaxParams));
};

bookStore.books.bookV2.getAsyncByIsbn = function(isbn, api_version,
ajaxParams) {
 var api_version = api_version ? api_version : '2.0';
 return abp.ajax($.extend(true, {
 url: abp.appPath + 'api/BookStore/Book/' + isbn + '' +
 abp.utils.buildQueryString([{ name: 'api-version', value: api_version }]) +
 '',
 type: 'GET'
 }, ajaxParams));
};

})();
```

```

Changing version manually

If an application service class supports multiple versions. You can inject '`ICurrentApiVersionInfo`' to switch versions in C#.

```

```cs
var currentApiVersionInfo =
 _abpApplication.ServiceProvider.GetRequiredService<ICurrentApiVersionInfo>();
var bookV4AppService =
 _abpApplication.ServiceProvider.GetRequiredService<IBookV4AppService>();

```

```

using (currentApiVersionInfo.Change(new
ApiVersionInfo(ParameterBindingSources.Query, "4.0")))
{
 book = await bookV4AppService.GetAsync();
 logger.LogWarning(book.Title);
 logger.LogWarning(book.ISBN);
}

using (currentApiVersionInfo.Change(new
ApiVersionInfo(ParameterBindingSources.Query, "4.1")))
{
 book = await bookV4AppService.GetAsync();
 logger.LogWarning(book.Title);
 logger.LogWarning(book.ISBN);
}
...

```

We have made a default version in the JS proxy. Of course, you can also manually change the version.

```

```js
bookStore.books.bookV4.get("4.0") // Manually change the version.
//Title: Mastering ABP Framework V4.0

bookStore.books.bookV4.get() // The latest supported version is used by
default.
//Title: Mastering ABP Framework V4.1
```
Auto API Controller

```cs
public override void PreConfigureServices(ServiceConfigurationContext
context)
{
    PreConfigure<AbpAspNetCoreMvcOptions>(options =>
    {
        //2.0 Version

        options.ConventionalControllers.Create(typeof(BookStoreWebAppModule).Assembly
, opts =>
        {
            opts.TypePredicate = t => t.Namespace ==
typeof(BookStore.Controllers.ConventionalControllers.v2.TodoAppService).Names
pace;
            opts.ApiVersions.Add(new ApiVersion(2, 0));
        });
        //1.0 Compatibility version

        options.ConventionalControllers.Create(typeof(BookStoreWebAppModule).Assembly
, opts =>
        {
            opts.TypePredicate = t => t.Namespace ==
typeof(BookStore.Controllers.ConventionalControllers.v1.TodoAppService).Names
pace;
            opts.ApiVersions.Add(new ApiVersion(1, 0));
        });
    });
}
```

```

```

 });
 });

public override void ConfigureServices(ServiceConfigurationContext context)
{
 var preActions =
context.Services.GetPreConfigureActions<AbpAspNetCoreMvcOptions>();
 Configure<AbpAspNetCoreMvcOptions>(options =>
{
 preActions.Configure(options);
});

context.Services.AddAbpApiVersioning(options =>
{
 // Show neutral/versionless APIs.
 options.UseApiBehavior = false;

 options.ReportApiVersions = true;
 options.AssumeDefaultVersionWhenUnspecified = true;

 options.ConfigureAbp(preActions.Configure());
});

Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.ChangeControllerModelApiExplorerGroupName = false;
});
}
```
## Swagger/VersionedApiExplorer
```cs
public override void ConfigureServices(ServiceConfigurationContext context)
{
 context.Services.AddAbpApiVersioning(options =>
{
 // Show neutral/versionless APIs.
 options.UseApiBehavior = false;

 options.ReportApiVersions = true;
 options.AssumeDefaultVersionWhenUnspecified = true;
});

context.Services.AddVersionedApiExplorer(
 options =>
{
 // add the versioned api explorer, which also adds
IApiVersionDescriptionProvider service
 // note: the specified format code will format the version as
"'v'major[.minor][-status]"
 options.GroupNameFormat = "'v'VVV";

 // note: this option is only necessary when versioning by url
segment. the SubstitutionFormat
}

```

```

 // can also be used to control the format of the API version in
route templates
 options.SubstituteApiVersionInUrl = true;
 });

 context.Services.AddTransient<IConfigureOptions<SwaggerGenOptions>,
ConfigureSwaggerOptions>();

 context.Services.AddAbpSwaggerGen(
 options =>
 {
 // add a custom operation filter which sets default values
 options.OperationFilter<SwaggerDefaultValues>();

 options.CustomSchemaIds(type => type.FullName);
 });

 Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.ChangeControllerModelApiExplorerGroupName = false;
});
}

public override void
OnApplicationInitialization(ApplicationInitializationContext context)
{
 var app = context.GetApplicationBuilder();
 var env = context.GetEnvironment();

 if (env.IsDevelopment())
 {
 app.UseDeveloperExceptionPage();
 }
 else
 {
 app.UseErrorPage();
 app.UseHsts();
 }

 app.UseHttpsRedirection();
 app.UseStaticFiles();
 app.UseRouting();
 app.UseAbpRequestLocalization();

 app.UseSwagger();
 app.UseSwaggerUI(
 options =>
 {
 var provider =
app.ApplicationServices.GetRequiredService<IApiVersionDescriptionProvider>();
 // build a swagger endpoint for each discovered API version
 foreach (var description in provider.ApiVersionDescriptions)
 {

options.SwaggerEndpoint($""/swagger/{description.GroupName}/swagger.json",
description.GroupName.ToUpperInvariant());
 }
 });
}

```

```
 app.UseConfiguredEndpoints();
 } ...

Custom multi-version API controller
```

ABP Framework will not affect to your APIs, you can freely implement your APIs according to the Microsoft's documentation.

Further information, see <https://github.com/dotnet/aspnet-api-versioning/wiki>

```
Sample source code
```

Follow the link below to get the sample's complete source-code

<https://github.com/abpframework/abp-samples/tree/master/Api-Versioning>

### 9.3 Auto API Controllers

```
Auto API Controllers
```

Once you create an [application service](../Application-Services.md), you generally want to create an API controller to expose this service as an HTTP (REST) API endpoint. A typical API controller does nothing but redirects method calls to the application service and configures the REST API using attributes like [HttpGet], [HttpPost], [Route]... etc.

ABP can \*\*automagically\*\* configure your application services as API Controllers by convention. Most of time you don't care about its detailed configuration, but it's possible to fully customize it.

```
Configuration
```

Basic configuration is simple. Just configure `AbpAspNetCoreMvcOptions` and use `ConventionalControllers.Create` method as shown below:

```
```csharp
[DependsOn(BookStoreApplicationModule)]
public class BookStoreWebModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        Configure<AbpAspNetCoreMvcOptions>(options =>
        {
            options
                .ConventionalControllers
                .Create(typeof(BookStoreApplicationModule).Assembly);
        });
    }
} ...
````
```

This example code configures all the application services in the assembly containing the class '[BookStoreApplicationModule](#)'. The figure below shows the resulting API on the [\[Swagger UI\]](#)(<https://swagger.io/tools/swagger-ui/>).

![bookstore-apis](../images/bookstore-apis.png)

### ### Examples

Some example method names and the corresponding routes calculated by convention:

| Service Method Name                                                                | HTTP Method | Route |
|------------------------------------------------------------------------------------|-------------|-------|
| GetAsync(Guid id)<br>/api/app/book/{id}                                            | GET         |       |
| GetListAsync()<br>/api/app/book                                                    | GET         |       |
| CreateAsync(CreateBookDto input)<br>/api/app/book                                  | POST        |       |
| UpdateAsync(Guid id, UpdateBookDto input)<br>/api/app/book/{id}                    | PUT         |       |
| DeleteAsync(Guid id)<br>/api/app/book/{id}                                         | DELETE      |       |
| GetEditorsAsync(Guid id)<br>/api/app/book/{id}/editors                             | GET         |       |
| CreateEditorAsync(Guid id, BookEditorCreateDto input)<br>/api/app/book/{id}/editor | POST        |       |

### ### HTTP Method

ABP uses a naming convention while determining the HTTP method for a service method (action):

- **Get**: Used if the method name starts with 'GetList', 'GetAll' or 'Get'.
- **Put**: Used if the method name starts with 'Put' or 'Update'.
- **Delete**: Used if the method name starts with 'Delete' or 'Remove'.
- **Post**: Used if the method name starts with 'Create', 'Add', 'Insert' or 'Post'.
- **Patch**: Used if the method name starts with 'Patch'.
- Otherwise, **Post** is used **by default**.

If you need to customize HTTP method for a particular method, then you can use one of the standard ASP.NET Core attributes (`[HttpPost]`, `[HttpGet]`, `[HttpPut]`... etc.). This requires to add [\[Microsoft.AspNetCore.Mvc.Core\]](#)(<https://www.nuget.org/packages/Microsoft.AspNetCore.Mvc.Core>) nuget package to your project that contains the service.

### ### Route

Route is calculated based on some conventions:

- \* It always starts with '\*\*/api\*\*'.
- \* Continues with a \*\*route path\*\*. Default value is '\*\*/app\*\*' and can be configured as like below:

```
```csharp
```

```

Configure<AbpAspNetCoreMvcOptions>(options =>
{
    options.ConventionalControllers
        .Create(typeof(BookStoreApplicationModule).Assembly, opts =>
    {
        opts.RootPath = "volosoft/book-store";
    });
});
```

```

Then the route for getting a book will be '\*\*/api/volosoft/book-store/book/{id}\*\*'. This sample uses two-level root path, but you generally use a single level depth.

- \* Continues with the \*\*normalized controller/service name\*\*. Normalization removes 'AppService', 'ApplicationService' and 'Service' postfixes and converts it to \*\*kebab-case\*\*. If your application service class name is 'ReadingBookAppService' then it becomes only '/reading-book'.
  - \* If you want to customize naming, then set the '[UrlControllerNameNormalizer](#)' option. It's a func delegate which allows you to determine the name per controller/service.
- \* If the method has an '\*\*id\*\*' parameter then it adds '\*\*/{id}\*\*' to the route.
- \* Then it adds the action name if necessary. Action name is obtained from the method name on the service and normalized by;
  - \* Removing '\*\*Async\*\*' postfix. If the method name is 'GetPhonesAsync' then it becomes 'GetPhones'.
  - \* Removing \*\*HTTP method prefix\*\*. 'GetList', 'GetAll', 'Get', 'Put', 'Update', 'Delete', 'Remove', 'Create', 'Add', 'Insert', 'Post' and 'Patch' prefixes are removed based on the selected HTTP method. So, 'GetPhones' becomes 'Phones' since 'Get' prefix is a duplicate for a GET request.
  - \* Converting the result to \*\*kebab-case\*\*.
  - \* If the resulting action name is \*\*empty\*\* then it's not added to the route. If it's not empty, it's added to the route (like '/phones'). For 'GetAllAsync' method name it will be empty, for 'GetPhonesAsync' method name it will be 'phones'.
  - \* Normalization can be customized by setting the '[UrlActionNameNormalizer](#)' option. It's an action delegate that is called for every method.
- \* If there is another parameter with 'Id' postfix, then it's also added to the route as the final route segment (like '/phoneId').

#### [#### Customizing the Route Calculation](#)

'[IConventionalRouteBuilder](#)' is used to build the route. It is implemented by the '[ConventionalRouteBuilder](#)' by default and works as explained above. You can replace/override this service to customize the route calculation strategy.

#### [#### Version 3.x Style Route Calculation](#)

The route calculation was different before the version 4.0. It was using camelCase conventions, while the ABP Framework version 4.0+ uses kebab-case. If you use the old route calculation strategy, follow one of the approaches;

- \* Set '[UseV3UrlStyle](#)' to '`true`' in the options of the '`options.ConventionalControllers.Create(...)`' method. Example:

```
```csharp
```

```
options.ConventionalControllers
    .Create(typeof(BookStoreApplicationModule).Assembly, opts =>
    {
        opts.UseV3UrlStyle = true;
    });
...;
```

This approach effects only the controllers for the `BookStoreApplicationModule`.

* Set `UseV3UrlStyle` to `true` for the `AbpConventionalControllerOptions` to set it globally. Example:

```
```csharp
Configure<AbpConventionalControllerOptions>(options =>
{
 options.UseV3UrlStyle = true;
});
```

Setting it globally effects all the modules in a modular application.

#### ## Service Selection

Creating conventional HTTP API controllers are not unique to application services actually.

#### ### IRemoteService Interface

If a class implements the `IRemoteService` interface then it's automatically selected to be a conventional API controller. Since application services inherently implement it, they are considered as natural API controllers.

#### ### RemoteService Attribute

`RemoteService` attribute can be used to mark a class as a remote service or disable for a particular class that inherently implements the `IRemoteService` interface. Example:

```
```csharp
[RemoteService(IsEnabled = false)] //or simply [RemoteService(false)]
public class PersonAppService : ApplicationService
{
}
```

TypePredicate Option

You can further filter classes to become an API controller by providing the `TypePredicate` option:

```
```csharp
services.Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.ConventionalControllers
 .Create(typeof(BookStoreApplicationModule).Assembly, opts =>
 {
```

```

 opts.TypePredicate = type => { return true; };
 });
}
```

```

Instead of returning '`true`' for every type, you can check it and return '`false`' if you don't want to expose this type as an API controller.

API Explorer

API Exploring a service that makes possible to investigate API structure by the clients. Swagger uses it to create a documentation and test UI for an endpoint.

API Explorer is automatically enabled for conventional HTTP API controllers by default. Use '`RemoteService`' attribute to control it per class or method level. Example:

```

```csharp
[RemoteService(IsMetadataEnabled = false)]
public class PersonAppService : ApplicationService
{
}
```

```

Disabled '`IsMetadataEnabled`' which hides this service from API explorer and it will not be discoverable. However, it still can be usable for the clients know the exact API path/route.

Replace or Remove Controllers.

In addition to [Overriding a Controller](./Customizing-Application-Modules-Overriding-Services.md#example-overriding-a-controller), you can also use a completely independent **Controller** to replace the controller in the framework or module.

They have the same `[route]`(<https://learn.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-7.0>), but can have **different** input and output parameters.

Replace built-in `AbpApplicationConfigurationController`

The '`ReplaceControllersAttribute`' indicates the replaced controller type.

```

```csharp
[ReplaceControllers(typeof(AbpApplicationConfigurationController))]
[Area("abp")]
[RemoteService(Name = "abp")]
public class ReplaceBuiltInController : AbpController
{
 [HttpGet("api/abp/application-configuration")]
 public virtual Task<MyApplicationConfigurationDto>
GetAsync(MyApplicationConfigurationRequestOptions options)
 {
 return Task.FromResult(new MyApplicationConfigurationDto());
 }
}
```

```

```

public class MyApplicationConfigurationRequestOptions : ApplicationConfigurationRequestOptions
{
}

public class MyApplicationConfigurationDto : ApplicationConfigurationDto
{
}
...
}

#### Remove controller

Configure `ControllersToRemove` of `AbpAspNetCoreMvcOptions` to remove the controllers.

```csharp
services.Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.ControllersToRemove.Add(typeof(AbpLanguagesController));
});
```

```

9.4 Dynamic C# API Clients

Dynamic C# API Client Proxies

ABP can dynamically create C# API client proxies to call your remote HTTP services (REST APIs). In this way, you don't need to deal with `'HttpClient'` and other low level details to call remote services and get results.

Dynamic C# proxies automatically handle the following stuff for you;

- * Maps C# **method calls** to remote server **HTTP calls** by considering the HTTP method, route, query string parameters, request payload and other details.
- * **Authenticates** the HTTP Client by adding access token to the HTTP header.
- * **Serializes** to and deserialize from JSON.
- * Handles HTTP API **versioning**.
- * Add **correlation id**, current **tenant** id and the current **culture** to the request.
- * Properly **handles the error messages** sent by the server and throws proper exceptions.

This system can be used by any type of .NET client to consume your HTTP APIs.

Static vs Dynamic Client Proxies

ABP provides **two types** of client proxy generation system. This document explains the **dynamic client proxies**, which generates client-side proxies on runtime. You can also see the [[Static C# API Client Proxies](#)](Static-CSharp-API-Clients.md) documentation to learn how to generate proxies on development time.

Development-time (static) client proxy generation has a **performance advantage** since it doesn't need to obtain the HTTP API definition on runtime. However, you should **re-generate** the client proxy code whenever you change your API endpoint definition. On the other hand, dynamic client proxies are generated on runtime and provides an **easier development experience**.

Service Interface

Your service/controller should implement an interface that is shared between the server and the client. So, first define a service interface in a shared library project, typically in the `Application.Contracts` project if you've created your solution using the startup templates.

Example:

```
```csharp
public interface IBookAppService : IApplicationService
{
 Task<List<BookDto>> GetListAsync();
}
...```

```

> Your interface should implement the `IRemoteService` interface to be automatically discovered. Since the `IApplicationService` inherits the `IRemoteService` interface, the `IBookAppService` above satisfies this condition.

Implement this class in your service application. You can use [auto API controller system](Auto-API-Controllers.md) to expose the service as a REST API endpoint.

## ## Client Proxy Generation

> The startup templates already comes pre-configured for the client proxy generation, in the `HttpApi.Client` project.

If you're not using a startup template, then execute the following command in the folder that contains the .csproj file of your client project:

```
...
abp add-package Volo.Abp.Http.Client
...```

```

> If you haven't done it yet, you first need to install the [ABP CLI](../CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.Http.Client).

Now, it's ready to create the client proxies. Example:

```
```csharp
[DependsOn(
    typeof(AbpHttpClientModule), //used to create client proxies
    typeof(BookStoreApplicationContractsModule) //contains the application
service interfaces
)]
public class MyClientAppModule : AbpModule
```

```

```

{
 public override void ConfigureServices(ServiceConfigurationContext
context)
 {
 //Create dynamic client proxies
 context.Services.AddHttpClientProxies(
 typeof(BookStoreApplicationContractsModule).Assembly
);
 }
}...

```

`AddHttpClientProxies` method gets an assembly, finds all service interfaces in the given assembly, creates and registers proxy classes.

### ### Endpoint Configuration

'RemoteServices' section in the `appsettings.json` file is used to get remote service address by default. The simplest configuration is shown below:

```

```json
{
    "RemoteServices": {
        "Default": {
            "BaseUrl": "http://localhost:53929/"
        }
    }
}...

```

See the "AbpRemoteServiceOptions" section below for more detailed configuration.

Usage

It's straightforward to use. Just inject the service interface in the client application code:

```

```csharp
public class MyService : ITransientDependency
{
 private readonly IBookAppService _bookService;

 public MyService(IBookAppService bookService)
 {
 _bookService = bookService;
 }

 public async Task DoIt()
 {
 var books = await _bookService.GetListAsync();
 foreach (var book in books)
 {
 Console.WriteLine($"[BOOK {book.Id}] Name={book.Name}");
 }
 }
}...

```

This sample injects the `IBookAppService` service interface defined above. The dynamic client proxy implementation makes an HTTP call whenever a service method is called by the client.

### ### IHttpclientProxy Interface

While you can inject `IBookAppService` like above to use the client proxy, you could inject `IHttpClientProxy<IBookAppService>` for a more explicit usage. In this case you will use the `Service` property of the `IHttpClientProxy<T>` interface.

### ## Configuration

#### ### AbpRemoteServiceOptions

`AbpRemoteServiceOptions` is automatically set from the `appsettings.json` by default. Alternatively, you can configure it in the `ConfigureServices` method of your [module](../Module-Development-Basics.md) to set or override it. Example:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    context.Services.Configure<AbpRemoteServiceOptions>(options =>
    {
        options.RemoteServices.Default =
            new RemoteServiceConfiguration("http://localhost:53929/");
    });

    //...
}
````
```

#### ### Multiple Remote Service Endpoints

The examples above have configured the "Default" remote service endpoint. You may have different endpoints for different services (as like in a microservice approach where each microservice has different endpoints). In this case, you can add other endpoints to your configuration file:

```
```json
{
    "RemoteServices": {
        "Default": {
            "BaseUrl": "http://localhost:53929/"
        },
        "BookStore": {
            "BaseUrl": "http://localhost:48392/"
        }
    }
}
````
```

`AddHttpClientProxies` method can get an additional parameter for the remote service name. Example:

```
```csharp
```

```
context.Services.AddHttpClientProxies(
    typeof(BookStoreApplicationContractsModule).Assembly,
    remoteServiceConfigurationName: "BookStore"
);
```

The `remoteServiceConfigurationName` parameter matches the service endpoint configured via `AbpRemoteServiceOptions`. If the `BookStore` endpoint is not defined then it fallbacks to the `Default` endpoint.

As Default Services

When you create a service proxy for `IBookAppService`, you can directly inject the `IBookAppService` to use the proxy client (as shown in the usage section). You can pass `asDefaultServices: false` to the `AddHttpClientProxies` method to disable this feature.

```
```csharp
context.Services.AddHttpClientProxies(
 typeof(BookStoreApplicationContractsModule).Assembly,
 asDefaultServices: false
);
```

Using `asDefaultServices: false` may only be needed if your application has already an implementation of the service and you do not want to override/replace the other implementation by your client proxy.

> If you disable `asDefaultServices`, you can only use `IHhttpClientProxy<T>` interface to use the client proxies. See the *\*IHhttpClientProxy Interface\** section above.

### ### Retry/Failure Logic & Polly Integration

If you want to add retry logic for the failing remote HTTP calls for the client proxies, you can configure the `AbpHttpClientBuilderOptions` in the `PreConfigureServices` method of your module class.

\*\*Example: Use the [Polly](https://github.com/App-vNext/Polly) library to retry 3 times on a failure\*\*

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
    PreConfigure<AbpHttpClientBuilderOptions>(options =>
    {
        options.ProxyClientBuildActions.Add((remoteServiceName,
clientBuilder) =>
        {
            clientBuilder.AddTransientHttpErrorPolicy(policyBuilder =>
                policyBuilder.WaitAndRetryAsync(
                    3,
                    i => TimeSpan.FromSeconds(Math.Pow(2, i))
                )
            );
        });
    });
}
```

```
} ...
```

This example uses the [\[Microsoft.Extensions.Http.Polly\]](#)(<https://www.nuget.org/packages/Microsoft.Extensions.Http.Polly>) package. You also need to import the `Polly` namespace (`using Polly;`) to be able to use the `WaitAndRetryAsync` method.

See Also

- * [\[Static C# Client Proxies\]](#)(Static-CSharp-API-Clients.md)

9.5 Integration Services

Integration Services

The **Integration Service** concept was created to distinguish the [\[application services\]](#)(Application-Services.md) that are built for inter-module (or inter-microservice) communication from the application services that are intended to be consumed from a UI or a client application.

The following figure shows a few microservices behind an API Gateway that is consumed by a UI application and 3rd-party client applications:

```
![integration-services](images/integration-services.png)
```

HTTP requests coming from out of the API Gateway can be called as **external request**, while the HTTP requests performed between microservices can be considered as **internal requests**. The application services that are designed to respond to these internal requests are called as **integration services**, because their purpose is to integrate microservices in the system, rather than respond to user requests.

Marking an Application Service as Integration Service

Assume that you have an application service named `ProductAppService`, and you want to use that application service as an integration service. In that case, you can use the `[IntegrationService]` attribute on top of the application service class as shown below:

```
```csharp
[IntegrationService]
public class ProductAppService : ApplicationService, IProductAppService
{
 // ...
}
...```

```

If your application service has an interface, like `IProductService` in this example, you can use it on the service interface:

```
```csharp
[IntegrationService]
public interface IProductAppService : IApplicationService
{
    // ...
}
...```

```

```
}

> If you've used the `[IntegrationService]` on top of your service interface, it is *not needed* to use on the service class too.
```

That's all. From now, ABP will handle your application service as integration service and implement the followings by convention:

- * That service is **not exposed** by default, unless you explicitly set `ExposeIntegrationServices` options (see the *Exposing Integration Services* section).
- * If you are using the [Auto API Controllers](API/Auto-API-Controllers.md) feature in your application, the **URL prefix** will be `/integration-api` instead of `/api` for your integration services. Thus, you can distinguish internal and external service communications and take additional actions, such as preventing REST API calls for integration services out of API Gateway.
- * **Audit logging** is disabled by default for the integration services. See the next section if you want to enable it.

Marking an MVC Controller as Integration Service

In addition to application services, you can mark a regular MVC Controller as integration service, using the same `IntegrationService` attribute, or inheriting an interface that has the `IntegrationService` attribute.

Example:

```
```csharp
[IntegrationService] // Mark as integration service
[Route("integration-api/products")]
public class ProductController : AbpControllerBase
{
 //...
}
```

```

When you use the `IntegrationService` attribute, ABP will handle your controller as integration service and implement the followings by convention:

- * That controller is **not exposed** to clients by default, unless you explicitly set `ExposeIntegrationServices` options (see the *Exposing Integration Services* section).
- * **Audit logging** is disabled by default for controller. See the next section if you want to enable it.

Configuration

Exposing Integration Services

Integration services and controllers are not exposed by default for security reasons. They typically don't require authorization, so you should **carefully and explicitly** allow them to be visible and usable to client applications.

To expose integration services and controllers, set `AbpAspNetCoreMvcOptions.ExposeIntegrationServices` to `true` in the

`ConfigureServices` method of your [module class](Module-Development-Basics.md):

```
```csharp
Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.ExposeIntegrationServices = true;
});
```

> Hiding integration services is useful when you are building reusable application modules, where they may be used in a monolith application or in a microservice system. In a monolith application, integration services don't need to be exposed outside since the modules may in-process communicate with each other. On the other hand, if you build a microservice solution and use that module as a service, it will be proper to expose the integration services, so other microservices can consume them remotely inside your private network (or Kubernetes cluster). In that case, be careful to not accidentally expose the integration services out of your private network. Configuring your API Gateway so that it blocks requests to `integration-api` prefixed URLs from outside of your network will be a good option.

### ### Enabling/Disabling the Audit Logging

Audit Logging is disabled by default for integration services but it can be enabled by configuring the `AbpAuditingOptions` [options class](Options.md) in the `ConfigureServices` method of your [module class](Module-Development-Basics.md):

```
```csharp
Configure<AbpAuditingOptions>(options =>
{
    options.IsEnabledForIntegrationService = true;
});
```

> Please refer to the [audit logging document](Audit-Logging.md) for other options and details.

Filtering Auto API Controllers

You can filter integration services (or non-integration services) while creating [Auto API Controllers](API/Auto-API-Controllers.md), using the `ApplicationServiceTypes` option of the `ConventionalControllerSetting` by configuring the `AbpAspNetCoreMvcOptions` as shown below:

```
```csharp
Configure<AbpAspNetCoreMvcOptions>(options =>
{
 options.ConventionalControllers.Create(
 typeof(MyApplicationModule).Assembly,
 conventionalControllerSetting =>
 {
 conventionalControllerSetting.ApplicationServiceTypes =
 ApplicationServiceTypes.IntegrationServices;
 });
});
```

Tip: You can call the `options.ConventionalControllers.Create` multiple times to configure regular application services and integration services with different options.

> Please refer to the [Auto API Controllers document](API/Auto-API-Controllers.md) for more information about the Auto API Controller system.

#### ## See Also

- \* [Application Services](Application-Services.md)
- \* [Auto API Controllers](API/Auto-API-Controllers.md)
- \* [Audit Logging](Audit-Logging.md)

## 9.6 Static C# API Clients

### # Static C# API Client Proxies

ABP can create C# API client proxy code to call your remote HTTP services (REST APIs). In this way, you don't need to deal with `HttpClient` and other low level details to call remote services and get results.

Static C# proxies automatically handle the following stuff for you;

- \* Maps C# \*\*method calls\*\* to remote server \*\*HTTP calls\*\* by considering the HTTP method, route, query string parameters, request payload and other details.
- \* \*\*Authenticates\*\* the HTTP Client by adding access token to the HTTP header.
- \* \*\*Serializes\*\* to and deserialize from JSON.
- \* Handles HTTP API \*\*versioning\*\*.
- \* Add \*\*correlation id\*\*, current \*\*tenant\*\* id and the current \*\*culture\*\* to the request.
- \* Properly \*\*handles the error messages\*\* sent by the server and throws proper exceptions.

This system can be used by any type of .NET client to consume your HTTP APIs.

### ## Static vs Dynamic Client Proxies

ABP provides \*\*two types\*\* of client proxy generation system. This document explains the \*\*static client proxies\*\*, which generates client-side code in your development time. You can also see the [Dynamic C# API Client Proxies](Dynamic-CSharp-API-Clients.md) documentation to learn how to use proxies generated on runtime.

Development-time (static) client proxy generation has a \*\*performance advantage\*\* since it doesn't need to obtain the HTTP API definition on runtime. However, you should \*\*re-generate\*\* the client proxy code whenever you change your API endpoint definition. On the other hand, dynamic client proxies are generated on runtime and provides an \*\*easier development experience\*\*.

### ## Service Interface

Your service/controller should implement an interface that is shared between the server and the client. So, first define a service interface in a shared library project, typically in the `'Application.Contracts'` project if you've created your solution using the startup templates.

Example:

```
```csharp
public interface IBookAppService : IApplicationService
{
    Task<List<BookDto>> GetListAsync();
}
...```

```

> Your interface should implement the `'IRemoteService'` interface to be automatically discovered. Since the `'IApplicationService'` inherits the `'IRemoteService'` interface, the `'IBookAppService'` above satisfies this condition.

Implement this class in your service application. You can use [auto API controller system](Auto-API-Controllers.md) to expose the service as a REST API endpoint.

With Contracts or Without Contracts

`'Without Contracts'` depending on target service's `'application.contracts'` package, so they can reuse the DTOs and other related classes. However, that can be a problem when we want to create fully independently developed and deployed microservices. We want to use the static proxy generation even without depending target service's `application.contracts` package.

`'With Contracts'` generate all the `'classes/enums/other'` types in the client side (including application service interfaces), This is also the default behavior of the `'generate-proxy'` command.

Client Proxy Generation

First, add `[Volo.Abp.Http.Client]`(<https://www.nuget.org/packages/Volo.Abp.Http.Client>) nuget package to your client project:

```
...
Install-Package Volo.Abp.Http.Client
...```

```

Then add `'AbpHttpClientModule'` dependency to your module:

```
```csharp
[DependsOn(typeof(AbpHttpClientModule))] //add the dependency
public class MyClientAppModule : AbpModule
{
}
...```

```

Now, it's ready to configure the application for the static client proxy generation.

#### ### With Contracts Example

```

```csharp
[DependsOn(
    typeof(AbpHttpClientModule) //used to create client proxies
)]
public class MyClientAppModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext
context)
    {
        // Prepare for static client proxy generation
        context.Services.AddStaticHttpClientProxies(
            typeof(MyClientAppModule).Assembly
        );
    }
}
```

```

#### ### Without Contracts Example

```

```csharp
[DependsOn(
    typeof(AbpHttpClientModule), //used to create client proxies
    typeof(BookStoreApplicationContractsModule) //contains the application
service interfaces
)]
public class MyClientAppModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext
context)
    {
        // Prepare for static client proxy generation
        context.Services.AddStaticHttpClientProxies(
            typeof(BookStoreApplicationContractsModule).Assembly
        );
    }
}
```

```

`AddStaticHttpClientProxies` method gets an assembly, finds all service interfaces in the given assembly, and prepares for static client proxy generation.

> The [application startup template](../Startup-Templates/Application.md) comes pre-configured for the \*\*dynamic\*\* client proxy generation, in the `HttpApi.Client` project. If you want to switch to the \*\*static\*\* client proxies, change `context.Services.AddHttpClientProxies` to `context.Services.AddStaticHttpClientProxies` in the module class of your `HttpApi.Client` project.

#### ### Endpoint Configuration

`RemoteServices` section in the `appsettings.json` file is used to get remote service address by default. The simplest configuration is shown below:

```

```json
{
    "RemoteServices": {

```

```
        "Default": {
            "BaseUrl": "http://localhost:53929/"
        }
    }
}
```

See the **AbpRemoteServiceOptions** section below for more detailed configuration.

Code Generation

Server side must be up and running while generating the client proxy code. So, run your application that serves the HTTP APIs on the `'BaseUrl'` that is configured like explained in the **Endpoint Configuration** section.

Open a command-line terminal in the root folder of your client project (`'.csproj'`) and type the following command:

With Contracts

```
```bash
abp generate-proxy -t csharp -u http://localhost:53929/
````
```

> If you haven't installed yet, you should install the [\[ABP CLI\]\(./CLI.md\)](#).

This command should generate the following files under the `'ClientProxies'` folder:

[!\[generated-static-client-proxies\]\(./images/generated-static-client-proxies-with-contracts.png\)](#)

- * `'BookClientProxy.Generated.cs'` is the actual generated proxy class in this example. `'BookClientProxy'` is a `'partial'` class where you can write your custom code (ABP won't override it).
- * `'IBookAppService.cs'` is the app service.
- * `'BookDto.cs'` is the Dto class which uses by app service.
- * `'app-generate-proxy.json'` contains information about the remote HTTP endpoint, so ABP can properly perform HTTP requests.

Without Contracts

```
```bash
abp generate-proxy -t csharp -u http://localhost:53929/ --without-contracts
````
```

This command should generate the following files under the `'ClientProxies'` folder:

[!\[generated-static-client-proxies\]\(./images/generated-static-client-proxies-without-contracts.png\)](#)

- * `'BookClientProxy.Generated.cs'` is the actual generated proxy class in this example. `'BookClientProxy'` is a `'partial'` class where you can write your custom code (ABP won't override it).
- * `'app-generate-proxy.json'` contains information about the remote HTTP endpoint, so ABP can properly perform HTTP requests.

> `generate-proxy` command generates proxies for only the APIs you've defined in your application. If you are developing a modular application, you can specify the `-m` (or `--module`) parameter to specify the module you want to generate proxies. See the `*generate-proxy*` section in the [ABP CLI](./CLI.md) documentation for other options.

Usage

It's straightforward to use the client proxies. Just inject the service interface in the client application code:

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IBookAppService _bookService;

 public MyService(IBookAppService bookService)
 {
 _bookService = bookService;
 }

 public async Task DoItAsync()
 {
 var books = await _bookService.GetListAsync();
 foreach (var book in books)
 {
 Console.WriteLine($"[BOOK {book.Id}] Name={book.Name}");
 }
 }
}...```

```

This sample injects the `IBookAppService` service interface defined above. The static client proxy implementation makes an HTTP call whenever a service method is called by the client.

#### ## Configuration

##### ### AbpRemoteServiceOptions

`'AbpRemoteServiceOptions'` is automatically set from the `'appsettings.json'` by default. Alternatively, you can configure it in the `'ConfigureServices'` method of your [module](./Module-Development-Basics.md) to set or override it. Example:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
    context.Services.Configure<AbpRemoteServiceOptions>(options =>
    {
        options.RemoteServices.Default =
            new RemoteServiceConfiguration("http://localhost:53929/");
    });

    //...
}
```

```

### ### Multiple Remote Service Endpoints

The examples above have configured the "Default" remote service endpoint. You may have different endpoints for different services (as like in a microservice approach where each microservice has different endpoints). In this case, you can add other endpoints to your configuration file:

```
```json
{
  "RemoteServices": {
    "Default": {
      "BaseUrl": "http://localhost:53929/"
    },
    "BookStore": {
      "BaseUrl": "http://localhost:48392/"
    }
  }
}...```

```

`AddStaticHttpClientProxies` method can get an additional parameter for the remote service name. Example:

```
```csharp
context.Services.AddStaticHttpClientProxies(
 typeof(BookStoreApplicationContractsModule).Assembly,
 remoteServiceConfigurationName: "BookStore"
)...
```

```

`remoteServiceConfigurationName` parameter matches the service endpoint configured via `AbpRemoteServiceOptions`. If the `BookStore` endpoint is not defined then it fallbacks to the `Default` endpoint.

Retry/Failure Logic & Polly Integration

If you want to add retry logic for the failing remote HTTP calls for the client proxies, you can configure the `AbpHttpClientBuilderOptions` in the `PreConfigureServices` method of your module class.

Example: Use the [Polly](https://github.com/App-vNext/Polly) library to retry 3 times on a failure

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext
context)
{
 PreConfigure<AbpHttpClientBuilderOptions>(options =>
 {
 options.ProxyClientBuildActions.Add((remoteServiceName,
clientBuilder) =>
 {
 clientBuilder.AddTransientHttpErrorPolicy(policyBuilder =>
 policyBuilder.WaitAndRetryAsync(
 3,
 i => TimeSpan.FromSeconds(Math.Pow(2, i))
)
 });
 });
}
```

```

```
        );
    });
}
}...
```

This example uses the [\[Microsoft.Extensions.Http.Polly\]](https://www.nuget.org/packages/Microsoft.Extensions.Http.Polly)(<https://www.nuget.org/packages/Microsoft.Extensions.Http.Polly>) package. You also need to import the `Polly` namespace (`using Polly;`) to be able to use the `'WaitAndRetryAsync'` method.

See Also

- * [\[Dynamic C# Client Proxies\]](#)(Dynamic-CSharp-API-Clients.md)

9.7 Swagger Integration

Swagger Integration

[\[Swagger \(OpenAPI\)\]](https://swagger.io/)(<https://swagger.io/>) is a language-agnostic specification for describing REST APIs. It allows both computers and humans to understand the capabilities of a REST API without direct access to the source code. Its main goals are to:

- Minimize the amount of work needed to connect decoupled services.
- Reduce the amount of time needed to accurately document a service.

ABP Framework offers a prebuilt module for full Swagger integration with small configurations.

Installation

> This package is already installed by default with the startup template. So, most of the time, you don't need to install it manually.

If installation is needed, it is suggested to use the [\[ABP CLI\]](#)(../CLI.md) to install this package.

Using the ABP CLI

Open a command line window in the folder of the `Web` or `HttpApi.Host` project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.Swashbuckle
````
```

> If you haven't done it yet, you first need to install the [\[ABP CLI\]](#)(../CLI.md). For other installation options, see [\[the package description page\]](#)(<https://abp.io/package-detail/Volo.Abp.Swashbuckle>).

Manual Installation

If you want to manually install;

1. Add the [\[Volo.Abp.Swashbuckle\]\(https://www.nuget.org/packages/Volo.Abp.Swashbuckle\)](https://www.nuget.org/packages/Volo.Abp.Swashbuckle) NuGet package to your 'Web' or 'HttpApi.Host' project:

```
'Install-Package Volo.Abp.Swashbuckle'
```

2. Add the 'AbpSwashbuckleModule' to the dependency list of your module:

```
```csharp
[DependsOn(
 //...other dependencies
 typeof(AbpSwashbuckleModule) // <-- Add module dependency like that
)]
public class YourModule : AbpModule
{
}
````
```

Configuration

First, we need to use 'AddAbpSwaggerGen' extension to configure Swagger in 'ConfigureServices' method of our module:

```
```csharp
public override void ConfigureServices(ServiceConfigurationContext context)
{
 var services = context.Services;

 //... other configurations.

 services.AddAbpSwaggerGen(
 options =>
 {
 options.SwaggerDoc("v1", new OpenApiInfo { Title = "Test API",
Version = "v1" });
 options.DocInclusionPredicate((docName, description) => true);
 options.CustomSchemaIds(type => type.FullName);
 }
);
````
```

Then we can use Swagger UI by calling 'UseAbpSwaggerUI' method in the 'OnApplicationInitialization' method of our module:

```
```csharp
public override void OnApplicationInitialization(ApplicationInitializationContext context)
{
 var app = context.GetApplicationBuilder();

 //... other configurations.

 app.UseAbpSwaggerUI(options =>
 {
 options.SwaggerEndpoint("/swagger/v1/swagger.json", "Test API");
 });
````
```

```
        //... other configurations.  
    }  
}
```

Hide ABP Endpoints on Swagger UI

If you want to hide ABP's default endpoints, call the `HideAbpEndpoints` method in your Swagger configuration as shown in the following example:

```
```csharp  
services.AddAbpSwaggerGen(
 options =>
 {
 //... other options

 //Hides ABP Related endpoints on Swagger UI
 options.HideAbpEndpoints();
 }
)
```

### ## Using Swagger with OAUTH

For non MVC/Tiered applications, we need to configure Swagger with OAUTH to handle authorization.

> ABP Framework uses OpenIddict by default. To get more information about OpenIddict, check this [\[documentation\]](#)(../Modules/OpenIddict.md).

To do that, we need to use `AddAbpSwaggerGenWithOAuth` extension to configure Swagger with OAuth issuer and scopes in `ConfigureServices` method of our module:

```
```csharp  
public override void ConfigureServices(ServiceConfigurationContext context)  
{  
    var services = context.Services;  
  
    //... other configarations.  
  
    services.AddAbpSwaggerGenWithOAuth(  
        "https://localhost:44341",           // authority issuer  
        new Dictionary<string, string>      //  
        {  
            {"Test", "Test API"}           // scopes  
        },  
        options =>  
        {  
            options.SwaggerDoc("v1", new OpenApiInfo { Title = "Test API",  
Version = "v1" });  
            options.DocInclusionPredicate((docName, description) => true);  
            options.CustomSchemaIds(type => type.FullName);  
        }  
    );  
}
```

Then we can use Swagger UI by calling `UseAbpSwaggerUI` method in the `OnApplicationInitialization` method of our module:

```
```csharp
public override void
OnApplicationInitialization(ApplicationInitializationContext context)
{
 var app = context.GetApplicationBuilder();

 //... other configurations.

 app.UseAbpSwaggerUI(options =>
 {
 options.SwaggerEndpoint("/swagger/v1/swagger.json", "Test API");

 var configuration =
context.ServiceProvider.GetRequiredService< IConfiguration>();
 options.OAuthClientId("Test_Swagger"); // clientId
 options.OAuthClientSecret("1q2w3e*"); // clientSecret
 });

 //... other configurations.
}
```

> Do not forget to set `OAuthClientId` and `OAuthClientSecret`.
```

Using Swagger with OIDC

You may also want to configure swagger using **OpenIdConnect** instead of OAUTH. This is especially useful when you need to configure different metadata address than the issuer in cases such as when you deploy your application to kubernetes cluster or docker. In these cases, metadata address will be used in sign-in process to reach the valid authentication server discovery endpoint over the internet and use the internal network to validate the obtained token.

To do that, we need to use `AddAbpSwaggerGenWithOidc` extension to configure Swagger with OAuth issuer and scopes in `ConfigureServices` method of our module:

```
```csharp
context.Services.AddAbpSwaggerGenWithOidc(
 configuration["AuthServer:Authority"],
 scopes: new[] { "SwaggerDemo" },
 // "authorization_code"
 flows: new[] { AbpSwaggerOidcFlows.AuthorizationCode },
 // When deployed on K8s, should be metadata URL of the reachable DNS over
internet like https://myauthserver.company.com
 discoveryEndpoint: configuration["AuthServer:Authority"],
 options =>
 {
 options.SwaggerDoc("v1", new OpenApiInfo { Title = "SwaggerDemo API",
Version = "v1" });
 options.DocInclusionPredicate((docName, description) => true);
 options.CustomSchemaIds(type => type.FullName);
 });
```

```

The '`flows`' is a list of default oidc flows that is supported by the oidc-provider (authserver). You can see the default supported flows below:

- `'AbpSwaggerOidcFlows.AuthorizationCode'`: The '`"authorization_code"`' flow is the **default and suggested** flow. **Doesn't require a client secret** when even there is a field for it.
- `'AbpSwaggerOidcFlows.Implicit'`: The deprecated '`"implicit"`' flow that was used for javascript applications.
- `'AbpSwaggerOidcFlows.Password'`: The legacy '`password`' flow which is also known as Resource Owner Password flow. You need to provide a user name, password and client secret for it.
- `'AbpSwaggerOidcFlows.ClientCredentials'`: The '`"client_credentials"`' flow that is used for server to server interactions.

You can define one or many flows which will be shown in the Authorize modal. You can set it `null` which will use the default "`authorization_code`" flow.

The '`discoveryEndpoint`' is the reachable openid-provider endpoint for the '`.well-known/openid-configuration`'. You can set it to `null` which will use default `AuthServer:Authority` appsettings configuration. If you are deploying your applications to a kubernetes cluster or docker swarm, you should set the '`discoveryEndpoint`' as real DNS that should be reachable over the internet.

> If are having problems with seeing the authorization modal, check the browser console logs and make sure you have a correct and reachable '`discoveryEndpoint`'

10 User Interface

10.1 MVC / Razor Pages

10.1.1 Overall

```
# ASP.NET Core MVC / Razor Pages UI
```

```
## Introduction
```

ABP Framework provides a convenient and comfortable way of creating web applications using the ASP.NET Core MVC / Razor Pages as the User Interface framework.

> ABP doesn't offer a new/custom way of UI development. You can continue to use your current skills to create the UI. However, it offers a lot of features to make your development easier and have a more maintainable code base.

```
### MVC vs Razor Pages
```

ASP.NET Core provides two models for UI development:

* **[MVC (Model-View-Controller)](<https://docs.microsoft.com/en-us/aspnet/core/mvc/>)** is the classic way that exists from the version 1.0. This model can be used to create UI pages/components and HTTP APIs.

* **[Razor Pages](<https://docs.microsoft.com/en-us/aspnet/core/razor-pages/>)** was introduced with the ASP.NET Core 2.0 as a new way to create web pages.

ABP Framework supports both of the MVC and the Razor Pages models. However, it is suggested to create the **UI pages with Razor Pages** approach and use the **MVC model to build HTTP APIs**. So, all the pre-build modules, samples and the documentation is based on the Razor Pages for the UI development, while you can always apply the MVC pattern to create your own pages.

Modularity

[Modularity]([../../Module-Development-Basics.md](#)) is one of the key goals of the ABP Framework. It is not different for the UI; It is possible to develop modular applications and reusable application modules with isolated and reusable UI pages and components.

The [application startup template]([../../Startup-Templates/Application.md](#)) comes with some application modules pre-installed. These modules have their own UI pages embedded into their own NuGet packages. You don't see their code in your solution, but they work as expected on runtime.

Theme System

ABP Framework provides a complete [Theming]([Theming.md](#)) system with the following goals:

- * Reusable [application modules]([../../Modules/Index.md](#)) are developed **theme-independent**, so they can work with any UI theme.
- * UI theme is **decided by the final application**.
- * The theme is distributed via NuGet/NPM packages, so it is **easily upgradable**.
- * The final application can **customize** the selected theme.

Current Themes

Currently, three themes are **officially provided**:

- * The [Basic Theme]([Basic-Theme.md](#)) is the minimalist theme with the plain Bootstrap style. It is **open source and free**.
- * The [Lepton Theme](<https://commercial.abp.io/themes>) is a **commercial** theme developed by the core ABP team and is a part of the [ABP Commercial](<https://commercial.abp.io/>) license.
- * The [LeptonX Theme](<https://x.leptontheme.com/>) is a theme that has both [commercial]([../../Themes/LeptonXLite/Commercial/Mvc.md](#)) and [lite]([../../Themes/LeptonXLite/AspNetCore.md](#)) choices.

There are also some community-driven themes for the ABP Framework (you can search on the web).

Base Libraries

There are a set of standard JavaScript/CSS libraries that comes pre-installed and supported by all the themes:

- [Twitter Bootstrap](<https://getbootstrap.com/>) as the fundamental HTML/CSS framework.

- [[JQuery](https://jquery.com/)] (<https://jquery.com/>) for DOM manipulation.
- [[DataTables.Net](https://datatables.net/)] (<https://datatables.net/>) for data grids.
- [[JQuery Validation](https://jqueryvalidation.org/)] (<https://jqueryvalidation.org/>) for client side & [[unobtrusive](https://github.com/aspnet/jquery-validation-unobtrusive)] (<https://github.com/aspnet/jquery-validation-unobtrusive>) validation
- [[FontAwesome](https://fontawesome.com/)] (<https://fontawesome.com/>) as the fundamental CSS font library.
- [[SweetAlert](https://sweetalert.js.org/)] (<https://sweetalert.js.org/>) to show fancy alert message and confirmation dialogs.
- [[Toastr](https://github.com/CodeSeven/toastr)] (<https://github.com/CodeSeven/toastr>) to show toast notifications.
- [[Lodash](https://lodash.com/)] (<https://lodash.com/>) as a utility library.
- [[Luxon](https://moment.github.io/luxon/)] (<https://moment.github.io/luxon/>) for date/time operations.
- [[JQuery Form](https://github.com/jquery/jquery-form)] (<https://github.com/jquery/jquery-form>) for AJAX forms.
- [[bootstrap-datepicker](https://github.com/uxsolutions/bootstrap-datepicker)] (<https://github.com/uxsolutions/bootstrap-datepicker>) to show date pickers.
- [[Select2](https://select2.org/)] (<https://select2.org/>) for better select/combo boxes.
- [[Timeago](http://timeago.yarp.com/)] (<http://timeago.yarp.com/>) to show automatically updating fuzzy timestamps.
- [[malihu-custom-scrollbar-plugin](https://github.com/malihu/malihu-custom-scrollbar-plugin)] (<https://github.com/malihu/malihu-custom-scrollbar-plugin>) for custom scrollbars.

You can use these libraries directly in your applications, without needing to manually import your page.

Layouts

The themes provide the standard layouts. So, you have responsive layouts with the standard features already implemented. The screenshot below has taken from the Application Layout of the [[Basic Theme](#)] (Basic-Theme.md) :

![basic-theme-application-layout] (../../images/basic-theme-application-layout.png)

See the [[Theming](#)] (Theming.md) document for more layout options and other details.

Layout Parts

A typical layout consists of multiple parts. The [[Theming](#)] (Theming.md) system provides [[menus](#)] (Navigation-Menu.md), [[toolbars](#)] (Toolbars.md), [[layout hooks](#)] (Layout-Hooks.md) and more to dynamically control the layout by your application and the modules you are using.

Features

This section highlights some of the features provided by the ABP Framework for the ASP.NET Core MVC / Razor Pages UI.

Dynamic JavaScript API Client Proxies

Dynamic JavaScript API Client Proxy system allows you to consume your server side HTTP APIs from your JavaScript client code, just like calling local functions.

****Example: Get a list of authors from the server****

```
```js
acme.bookStore.authors.author.getList({
 maxResultCount: 10
})
```

```
}).then(function(result) {
 console.log(result.items);
});
```

`acme.bookStore.authors.author.getList` is an auto-generated function that internally makes an AJAX call to the server.

See the [Dynamic JavaScript API Client Proxies] (Dynamic-JavaScript-Proxies.md) document for more.

### ### Bootstrap Tag Helpers

ABP makes it easier & type safe to write Bootstrap HTML.

#### **\*\*Example: Render a Bootstrap modal\*\***

```
```html
<abp-modal>
  <abp-modal-header title="Modal title" />
  <abp-modal-body>
    Woohoo, you're reading this text in a modal!
  </abp-modal-body>
  <abp-modal-footer buttons="@({Save, Close})" />
</abp-modal>
```

See the [Tag Helpers] (Tag-Helpers/Index.md) document for more.

Forms & Validation

ABP provides `abp-dynamic-form` and `abp-input` tag helpers to dramatically simplify to create a fully functional form that automates localization, validation and AJAX submission.

****Example: Use `abp-dynamic-form` to create a complete form based on a model****

```
```html
<abp-dynamic-form abp-model="Movie" submit-button="true" />
```

See the [Forms & Validation] (Forms-Validation.md) document for details.

### ### Bundling & Minification / Client Side Libraries

ABP provides a flexible and modular Bundling & Minification system to create bundles and minify style/script files on runtime.

```
```html
<abp-style-bundle>
  <abp-style src="/libs/bootstrap/css/bootstrap.css" />
```

```
<abp-style src="/libs/font-awesome/css/font-awesome.css" />
<abp-style src="/libs/toastr/toastr.css" />
<abp-style src="/styles/my-global-style.css" />
</abp-style-bundle>
````
```

Also, Client Side Package Management system offers a modular and consistent way of managing 3rd-party library dependencies.

See the [\[Bundling & Minification\]](#) (`Bundling-Minification.md`) and [\[Client Side Package Management\]](#) (`Client-Side-Package-Management.md`) documents.

#### ### JavaScript APIs

[\[JavaScript APIs\]](#) (`JavaScript-API/Index.md`) provides a strong abstractions to the server side localization, settings, permissions, features... etc. They also provide a simple way to show messages and **\*\*notifications\*\*** to the user.

#### ### Modals, Alerts, Widgets and More

ABP Framework provides a lot of built-in solutions to common application requirements;

- \* [\[Widget System\]](#) (`Widgets.md`) can be used to create reusable widgets & create dashboards.
- \* [\[Page Alerts\]](#) (`Page-Alerts.md`) makes it easy to show alerts to the user.
- \* [\[Modal Manager\]](#) (`Modals.md`) provides a simple way to build and use modals.
- \* [\[Data Tables\]](#) (`Data-Tables.md`) integration makes straightforward to create data grids.

#### ## Customization

There are a lot of ways to customize the theme and the UIs of the pre-built modules. You can override components, pages, static resources, bundles and more. See the [\[User Interface Customization Guide\]](#) (`Customization-User-Interface.md`).

### 10.1.2 Navigation / Menus

#### # ASP.NET Core MVC / Razor Pages UI: Navigation Menu

Every application has a main menu to allow users to navigate to pages/screens of the application. Some applications may contain more than one menu in different sections of the UI.

ABP Framework is a [\[modular\]](#) (`../Module-Development-Basics.md`) application development framework. **\*\*Every module may need to add items to the menu\*\***.

So, ABP Framework **\*\*provides a menu infrastructure\*\*** where;

- \* The application or the modules can add items to a menu, without knowing how the menu is rendered.
- \* The [\[theme\]](#) (`Theming.md`) properly renders the menu.

## ## Adding Menu Items

In order to add menu items (or manipulate the existing items) you need to create a class implementing the `IMenuContributor` interface.

➤ The [application startup template](../../../../Startup-Templates/Application.md) already contains an implementation of the `IMenuContributor`. So, you can add items inside that class instead of creating a new one.

**\*\*Example: Add a \*CRM\* menu item with \*Customers\* and \*Orders\* sub menu items\*\***

```
```csharp
using System.Threading.Tasks;
using MyProject.Localization;
using Volo.Abp.UI.Navigation;

namespace MyProject.Web.Menus
{
    public class MyProjectMenuContributor : IMenuContributor
    {
        public async Task ConfigureMenuAsync(MenuConfigurationContext context)
        {
            if (context.Menu.Name == StandardMenus.Main)
            {
                await ConfigureMainMenuAsync(context);
            }
        }

        private async Task ConfigureMainMenuAsync(MenuConfigurationContext context)
        {
            var l = context.GetLocalizer<MyProjectResource>();

            context.Menu.AddItem(
                new ApplicationMenuItem("MyProject.Crm", l["Menu:CRM"])
                    .AddItem(new ApplicationMenuItem(
                        name: "MyProject.Crm.Customers",
                        displayName: l["Menu:Customers"],
                        url: "/crm/customers"))
                    .AddItem(new ApplicationMenuItem(
                        name: "MyProject.Crm.Orders",
                        displayName: l["Menu:Orders"],
                        url: "/crm/orders"))
            );
        }
    }
}```
```

* This example adds items only to the main menu (`StandardMenus.Main`: see the **Standard Menus** section below).

- * It gets a `IStringLocalizer` from `context` to [localize](../../Localization.md) the display names of the menu items.
- * Adds the Customers and Orders as children of the CRM menu.

Once you create a menu contributor, you need to add it to the `AbpNavigationOptions` in the `ConfigureServices` method of your module:

```
```csharp
Configure<AbpNavigationOptions>(options =>
{
 options.MenuContributors.Add(new MyProjectMenuContributor());
});```

```

This example uses some localization keys as display names those should be defined in the localization file:

```
```json
"Menu:CRM": "CRM",
"Menu:Orders": "Orders",
"Menu:Customers": "Customers"
```

```

See the [localization document](../../Localization.md) to learn more about the localization.

When you run the application, you will see the menu items added to the main menu:

! [nav-main-menu] ../../images/nav-main-menu.png)

➤ The menu is rendered by the current UI theme. So, the look of the main menu can be completely different based on your theme.

Here, a few notes on the menu contributors;

- \* ABP Framework calls the `ConfigureMenuAsync` method **\*\*whenever need to render\*\*** the menu.
- \* Every menu item can have **\*\*children\*\***. So, you can add menu items with **\*\*unlimited depth\*\*** (however, your UI theme may not support unlimited depth).
- \* Only leaf menu items have `url`'s normally. When you click to a parent menu, its sub menu is opened or closed, you don't navigate the `url` of a parent menu item.
- \* If a menu item has no children and has no `url` defined, then it is not rendered on the UI. This simplifies to authorize the menu items: You only authorize the child items (see the next section). If none of the children are authorized, then the parent automatically disappears.

### ### Menu Item Properties

There are more options of a menu item (the constructor of the `ApplicationMenuItem` class). Here, the list of all available options;

- \* `name` (`string`, required): The **\*\*unique name\*\*** of the menu item.

- \* `displayName` (`string`, required): Display name/text of the menu item. You can [localize](../../Localization.md) this as shown before.
- \* `url` (`string`): The URL of the menu item.
- \* `icon` (`string`): An icon name. Free [Font Awesome](https://fontawesome.com/) icon classes are supported out of the box. Example: `fa fa-book`. You can use any CSS font icon class as long as you include the necessary CSS files to your application.
- \* `order` (`int`): The order of the menu item. Default value is `1000`. Items are sorted by the adding order unless you specify an order value.
- \* `customData` (`Dictionary<string, object>`): A dictionary that allows storing custom objects that you can associate with the menu item and use it while rendering the menu item.
- \* `target` (`string`): Target of the menu item. Can be `null` (default), `"\_blank"`, `"\_self"`, `"\_parent"`, `"\_top"` or a frame name for web applications.
- \* `elementId` (`string`): Can be used to render the element with a specific HTML `id` attribute.
- \* `cssClass` (`string`): Additional string classes for the menu item.
- \* `groupName` (`string`): Can be used to group menu items.

### ### Authorization

As seen above, a menu contributor contributes to the menu dynamically. So, you can perform any custom logic or get menu items from any source.

One use case is the [authorization](../../Authorization.md). You typically want to add menu items by checking a permission.

#### **\*\*Example: Check if the current user has a permission\*\***

```
```csharp
if (await context.IsGrantedAsync("MyPermissionName"))
{
    //...add menu items
}
```

```

For the authorization, you can use `RequirePermissions` extension method as a shortcut. It is also more performant, ABP optimizes the permission check for all the items.

```
```csharp
context.Menu.AddItem(
    new ApplicationMenuItem("MyProject.Crm", 1["Menu:CRM"])
        .AddItem(new ApplicationMenuItem(
            name: "MyProject.Crm.Customers",
            displayName: 1["Menu:Customers"],
            url: "/crm/customers")
            .RequirePermissions("MyProject.Crm.Customers"))
        .AddItem(new ApplicationMenuItem(
            name: "MyProject.Crm.Orders",
            displayName: 1["Menu:Orders"],
            url: "/crm/orders")
            .RequirePermissions("MyProject.Crm.Orders"))
)
```

```

```
);
```
```

> You can use `context.AuthorizationService` to directly access to the `IAuthorizationService`.

Resolving Dependencies

`context.ServiceProvider` can be used to resolve any service dependency.

****Example: Get a service****

```
~~~~csharp  
var myService = context.ServiceProvider.GetRequiredService<IMyService>();  
//...use the service  
~~~~
```

> You don't need to care about releasing/disposing services. ABP Framework handles it.

The Administration Menu

There is a special menu item in the menu menu that is added by the ABP Framework: The ***Administration*** menu. It is typically used by the pre-built admin [application modules](../../Modules/Index.md) :

![nav-main-menu-administration](../../images/nav-main-menu-administration.png)

If you want to add menu items under the ***Administration*** menu item, you can use the `context.Menu.GetAdministration()` extension method:

```
~~~~csharp  
context.Menu.GetAdministration().AddItem(...)  
~~~~
```

Manipulating the Existing Menu Items

ABP Framework executes the menu contributors by the [module dependency order](../../Module-Development-Basics.md). So, you can manipulate the menu items that your application or module (directly or indirectly) depends on.

****Example: Set an icon for the `Users` menu item added by the [Identity Module](../../Modules/Identity.md)****

```
~~~~csharp  
var userMenu = context.Menu.FindMenuItem(IdentityMenuNames.Users);  
userMenu.Icon = "fa fa-users";  
~~~~
```

> `context.Menu` gives you ability to access to all the menu items those have been added by the previous menu contributors.

Menu Groups

You can define groups and associate menu items with a group.

Example:

```
```csharp
using System.Threading.Tasks;
using MyProject.Localization;
using Volo.Abp.UI.Navigation;

namespace MyProject.Web.Menus
{
 public class MyProjectMenuContributor : IMenuContributor
 {
 public async Task ConfigureMenuAsync(MenuConfigurationContext context)
 {
 if (context.Menu.Name == StandardMenus.Main)
 {
 await ConfigureMainMenuAsync(context);
 }
 }

 private async Task ConfigureMainMenuAsync(MenuConfigurationContext context)
 {
 var l = context.GetLocalizer<MyProjectResource>();

 context.Menu.AddGroup(
 new ApplicationMenuGroup(
 name: "Main",
 displayName: l["Main"])
)
 }

 context.Menu.AddItem(
 new ApplicationMenuItem("MyProject.Crm", l["Menu:CRM"], groupName: "Main")
 .AddItem(new ApplicationMenuItem(
 name: "MyProject.Crm.Customers",
 displayName: l["Menu:Customers"],
 url: "/crm/customers"))
 .AddItem(new ApplicationMenuItem(
 name: "MyProject.Crm.Orders",
 displayName: l["Menu:Orders"],
 url: "/crm/orders"))
);
 }
}
```

```

➤ The UI theme will decide whether to render the groups or not, and if it decides to render, the way it's rendered is up to the theme. Only the LeptonX theme implements the menu group.

Standard Menus

A menu is a ****named**** component. An application may contain more than one menus with different, unique names. There are two pre-defined standard menus:

- * `Main` : The main menu of the application. Contains links to the page of the application. Defined as a constant: `Volo.Abp.UI.Navigation.StandardMenus.Main`.
- * `User` : User profile menu. Defined as a constant: `Volo.Abp.UI.Navigation.StandardMenus.User`.

The `Main` menu already covered above. The `User` menu is available when a user has logged in:

```
![user-menu](../../images/user-menu.png)
```

You can add items to the `User` menu by checking the `context.Menu.Name` as shown below:

```
```csharp
if (context.Menu.Name == StandardMenus.User)
{
 //...add items
}
````
```

IMenuManager

`IMenuManager` is generally used by the UI [theme](Theming.md) to render the menu items on the UI. So, ****you generally don't need to directly use**** the `IMenuManager`.

****Example: Getting the `Main` menu items****

```
```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.RazorPages;
using Volo.Abp.UI.Navigation;

namespace MyProject.Web.Pages
{
 public class IndexModel : PageModel
 {
 private readonly IMenuManager _menuManager;

 public IndexModel(IMenuManager menuManager)
 {
 _menuManager = menuManager;
 }

 public async Task OnGetAsync()
 {
 var mainMenu = await _menuManager.GetAsync(StandardMenus.Main);
 }
 }
}
```

```

 foreach (var menuItem in mainMenu.Items)
 {
 //...
 }
 }
}..

```

### 10.1.3 Forms & Validation

#### # ASP.NET Core MVC / Razor Pages: Forms & Validation

ABP Framework provides infrastructure and conventions to make easier to create forms, localize display names for the form elements and handle server & client side validation;

- \* [\[abp-dynamic-form\]](#) (Tag-Helpers/Dynamic-Forms.md) tag helper automates **\*\*creating a complete form\*\*** from a C# model class: Creates the input elements, handles localization and client side validation.
- \* [\[ABP Form tag helpers\]](#) (Tag-Helpers/Form-elements.md) (`abp-input`, `abp-select`, `abp-radio`...) render **\*\*a single form element\*\*** with handling localization and client side validation.
- \* ABP Framework automatically **\*\*localizes the display name\*\*** of a form element without needing to add a `[DisplayName]` attribute.
- \* **\*\*Validation errors\*\*** are automatically localized based on the user culture.

➤ This document is for the **\*\*client side validation\*\*** and it doesn't cover the server side validation. Check the [\[validation document\]\(../../../../Validation.md\)](#) for server side validation infrastructure.

#### ## The Classic Way

In a typical Bootstrap based ASP.NET Core MVC / Razor Pages UI, you [\[need to write\]\(https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation#client-side-validation\)](#) such a boilerplate code to create a simple form element:

```

~~~~html
<div class="form-group">
    <label asp-for="Movie.ReleaseDate" class="control-label"></label>
    <input asp-for="Movie.ReleaseDate" class="form-control" />
    <span asp-validation-for="Movie.ReleaseDate" class="text-danger"></span>
</div>
~~~~

```

You can continue to use this approach if you need or prefer it. However, ABP Form tag helpers can produce the same output with a minimal code.

#### ## ABP Dynamic Forms

[[abp-dynamic-form](#)] ([TagHelpers/Dynamic-Forms.md](#)) tag helper completely automates the form creation. Take this model class as an example:

```
```csharp
using System;
using System.ComponentModel.DataAnnotations;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Form;

namespace MyProject.Web.Pages
{
    public class MovieViewModel
    {
        [Required]
        [StringLength(256)]
        public string Name { get; set; }

        [Required]
        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }

        [Required]
        [TextArea]
        [StringLength(1000)]
        public string Description { get; set; }

        public Genre Genre { get; set; }

        public float? Price { get; set; }

        public bool PreOrder { get; set; }
    }
}
```

```

It uses the data annotation attributes to define validation rules and UI styles for the properties. `Genre` , is an `enum` in this example:

```
```csharp
namespace MyProject.Web.Pages
{
    public enum Genre
    {
        Classic,
        Action,
        Fiction,
        Fantasy,
        Animation
    }
}
```

```

In order to create the form in a razor page, create a property in your `PageModel` class:

```

```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace MyProject.Web.Pages
{
    public class CreateMovieModel : PageModel
    {
        [BindProperty]
        public MovieViewModel Movie { get; set; }

        public void OnGet()
        {
            Movie = new MovieViewModel();
        }

        public async Task OnPostAsync()
        {
            if (ModelState.IsValid)
            {
                //TODO: Save the Movie
            }
        }
    }
}
```

```

Then you can render the form in the `\*.cshtml` file:

```

```html
@page
@model MyProject.Web.Pages.CreateMovieModel

<h2>Create a new Movie</h2>

<abp-dynamic-form abp-model="Movie" submit-button="true" />
```

```

The result is shown below:

`![abp-dynamic-form-result](../../images/abp-dynamic-form-result.png)`

See the *\*Localization & Validation\** section below to localize the field display names and see how the validation works.

> See [[its own document](#)] (Tag-Helpers/Dynamic-Forms.md) for all options of the `abp-dynamic-form` tag helper.

## ABP Form Tag Helpers

``abp-dynamic-form`` covers most of the scenarios and allows you to control and customize the form using the attributes.

However, if you want to **\*\*render the form body yourself\*\*** (for example, you may want to fully control the **\*\*form layout\*\***), you can directly use the [ABP Form Tag Helpers](Tag-Helpers/Form-elements.md). The same auto-generated form above can be created using the ABP Form Tag Helpers as shown below:

```
```html
@page
@model MyProject.Web.Pages.CreateMovieModel

<h2>Create a new Movie</h2>

<form method="post">
    <abp-input asp-for="Movie.Name"/>
    <abp-input asp-for="Movie.ReleaseDate"/>
    <abp-input asp-for="Movie.Description"/>
    <abp-select asp-for="Movie.Genre"/>
    <abp-input asp-for="Movie.Price"/>
    <abp-input asp-for="Movie.PreOrder"/>
    <abp-button button-type="Primary" type="submit">Save</abp-button>
</form>
```

```

➤ See the [ABP Form Tag Helpers](Tag-Helpers/Form-elements.md) document for details of these tag helpers and their options.

## ## Validation & Localization

Both of the Dynamic Form and the Form Tag Helpers **\*\*automatically validate\*\*** the input based on the data annotation attributes and shows validation error messages on the user interface. Error messages are **\*\*automatically localized\*\*** based on the current culture.

### \*\*Example: User leaves empty a required string property\*\*

![`abp-form-validation-error`](../../images/abp-form-validation-error.png)

The error message below is shown if the language is French:

![`abp-form-validation-error`](../../images/abp-form-validation-error-french.png)

Validation errors are already [translated](#) (<https://github.com/abpframework/abp/tree/dev/framework/src/Volo.Abp.Validation/Volo/Abp/Validation/Localization>) a lot of languages. You can [contribute](#) (../../Contribution/Index.md) to the translation for your own language or override the texts for your own application by following the [localization](#) (../../Localization.md) documentation.

## ## Display Name Localization

ABP Framework uses the property name as the field name on the user interface. You typically want to `[localize](../../Localization.md)` this name based on the current culture.

ABP Framework can conventionally localize the fields on the UI when you add the localization keys to the localization JSON files.

Example: French localization for the `*Name*` property (add into the ``fr.json`` in the application):

```
```js
"Name": "Nom"
```
```

Then the UI will use the given name for French language:

```
![abp-form-input-validation-error](../../images/abp-form-input-validation-error-french-name.png)
```

#### ### Using the ``DisplayName:`` Prefix

Directly using the property name as the localization key may be a problem if you need to use the property name for other purpose, which a different translation value. In this case, use the ``DisplayName:`` prefix for the localization key:

```
```js
"DisplayName:Name": "Nom"
```
```

ABP prefers to use the ``DisplayName:Name`` key over the ``Name`` key if it does exists.

#### ### Using a Custom Localization Key

If you need, you can use the ``[DisplayName]`` attribute to specify the localization key for a specific property:

```
```csharp
[DisplayName("MyNameKey")]
public string Name { get; set; }
```
```

In this case, you can add an entry to the localization file using the key ``MyNameKey``.

> If you use the ``[DisplayName]`` but not add a corresponding entity to the localization file, then ABP Framework shows the given key as the field name, ``MyNameKey`` for this case. So, it provides a way to specify a hard coded display name even if you don't need to use the localization system.

#### ### Enum Localization

Enum members are also automatically localized wherever possible. For example, when we added ``<abp-select asp-for="Movie.Genre"/>`` to the form (like we did in the *ABP Form Tag*

*Helpers*\* section), ABP can automatically fill the localized names of Enum members. To enable it, you should define the localized values in your localization JSON file. Example entries for the `Genre` Enum defined in the \*ABP Form Tag Helpers\* section:

```
```json
"Enum:Genre.0": "Classic movie",
"Enum:Genre.1": "Action movie",
"Enum:Genre.2": "Fiction",
"Enum:Genre.3": "Fantasy",
"Enum:Genre.4": "Animation/Cartoon"
````
```

You can use one of the following syntaxes for the localization keys:

- \* `Enum:<enum-type-name>.<enum-value>`
- \* `<enum-type-name>.<enum-value>`

> Remember that if you don't specify values for your Enum, the values will be ordered, starting from `0`.

> MVC tag helpers also support using Enum member names instead of values (so, you can define `Enum:Genre.Action` instead of `Enum:Genre.1`, for example), but it is not suggested. Because, when you serialize Enum properties to JSON and send to clients, default serializer uses Enum values instead of Enum names. So, the Enum name won't be available to clients, and it will be a problem if you want to use the same localization values on the client side.

#### ## See Also

- \* [Server Side Validation](../../Validation.md)

### 10.1.4 Modals

#### # ASP.NET Core MVC / Razor Pages UI: Modals

While you can continue to use the standard [Bootstrap way](<https://getbootstrap.com/docs/4.5/components/modal/>) to create, open and manage modals in your applications, ABP Framework provides a **flexible** way to manage modals by **automating common tasks** for you.

##### **Example: A modal dialog to create a new role entity**

! [modal-manager-example-modal](../../images/modal-manager-example-modal.png)

ABP Framework provides the following benefits for such a modal with a form inside it;

- \* **Lazy loads** the modal HTML into the page and **removes** it from the DOM once its closed. This makes easy to consume a reusable modal dialog. Also, every time you open the modal, it will be a fresh new modal, so you don't have to deal with resetting the modal content.

- \* **Auto-focuses** the first input of the form once the modal has been opened. You can also specify it using a `function` or `jquery selector`.
- \* Automatically determines the **form** inside a modal and posts the form via **AJAX** instead of normal page post.
- \* Automatically checks if the form inside the modal **has changed, but not saved**. It warns the user in this case.
- \* Automatically **disables the modal buttons** (save & cancel) until the AJAX operation completes.
- \* Makes it easy to register a **JavaScript object that is initialized** once the modal has loaded.

So, it makes you write less code when you deal with the modals, especially the modals with a form inside.

## ## Basic Usage

### ### Creating a Modal as a Razor Page

To demonstrate the usage, we are creating a simple Razor Page, named `ProductInfoModal.cshtml`, under the `/Pages/Products` folder:

```
![modal-page-on-rider] (../../images/modal-page-on-rider.png)
```

#### **ProductInfoModal.cshtml Content:**

```
~~~~html
@page
@model MyProject.Web.Pages.Products.ProductInfoModalModel
 @{
    Layout = null;
}
<abp-modal>
    <abp-modal-header title="Product Information"></abp-modal-header>
    <abp-modal-body>
        <h3>@Model.ProductName</h3>
        <div>
            
        </div>
        <p>
            @Model.ProductDescription
        </p>
        <p>
            <small><i>Reference: https://acme.com/catalog/</i></small>
        </p>
    </abp-modal-body>
    <abp-modal-footer buttons="Close"></abp-modal-footer>
</abp-modal>
~~~~
```

\* This page sets the `Layout` to `null` since we will show this as a modal. So, no need to wrap with a layout.

\* It uses [abp-modal tag helper] (Tag-Helpers/Modals.md) to simplify creating the modal HTML code. You can use the standard Bootstrap modal code if you prefer it.

#### \*\*ProductInfoModalModel.cshtml.cs Content:\*\*

```
```csharp
using Volo.Abp.AspNetCore.Mvc.UI.RazorPages;

namespace MyProject.Web.Pages.Products
{
    public class ProductInfoModalModel : AbpPageModel
    {
        public string ProductName { get; set; }

        public string ProductDescription { get; set; }

        public string ProductImageUrl { get; set; }

        public void OnGet()
        {
            ProductName = "Acme Indestructo Steel Ball";
            ProductDescription = "The ACME Indestructo Steel Ball is completely indestructible, there is nothing that can destroy it!";
            ProductImageUrl = "https://acme.com/catalog/acmeindestructo.jpg";
        }
    }
}
```

```

You can surely get the product info from a database or API. We are setting the properties hard-coded for the sake of simplicity,

#### ### Defining the Modal Manager

Once you have a modal, you can open it in any page using some simple **\*\*JavaScript\*\*** code.

First, create an `abp.ModalManager` object by setting the `viewUrl`, in the JavaScript file of the page that will use the modal:

```
```js
var productInfoModal = new abp.ModalManager({
    viewUrl: '/Products/ProductInfoModal'
});
```

```

> If you only need to specify the `viewUrl`, you can directly pass it to the `ModalManager` constructor, as a shortcut. Example: `new abp.ModalManager('/Products/ProductInfoModal');`

#### ### Opening the Modal

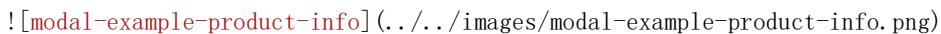
Then open the modal whenever you need:

```
```js
productInfoModal.open();
````
```

You typically want to open the modal when something happens; For example, when the user clicks a button:

```
```js
$( '#openProductInfoModal' ).click(function() {
    productInfoModal.open();
});
````
```

The resulting modal will be like that:



#### #### Opening the Modal with Arguments

When you call the `open()` method, `ModalManager` loads the modal HTML by requesting it from the `viewUrl`. You can pass some **query string parameters** to this URL when you open the modal.

#### **\*\*Example: Pass the product id while opening the modal\*\***

```
```js
productInfoModal.open({
    productId: 42
});
````
```

You can add a `productId` parameter to the get method:

```
```csharp
using Volo.Abp.AspNetCore.Mvc.UI.RazorPages;

namespace MyProject.Web.Pages.Products
{
    public class ProductInfoModalModel : AbpPageModel
    {
        //...

        public async Task OnGetAsync(int productId) //Add productId parameter
        {
            //TODO: Get the product with database with the given productId
            //...
        }
    }
}
````
```

In this way, you can use the `productId` to query the product from a data source.

## ## Modals with Forms

`abp.ModalManager` handles various common tasks (described in the introduction) when you want to use a form inside the modal.

### #### Example Modal with a Form

This section shows an example form to create a new product.

#### ##### Creating the Razor Page

For this example, creating a new Razor Page, named `ProductCreateModal.cshtml`, under the `/Pages/Products` folder:

```
! [product-create-modal-page-on-rider] (././images/product-create-modal-page-on-rider.png)
```

#### \*\*ProductCreateModal.cshtml Content:\*\*

```
```html
@page
@using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Modal
@model MyProject.Web.Pages.Products.ProductCreateModalModel
 @{
    Layout = null;
}
<form method="post" action="@Url.Page("/Products/ProductCreateModal")">
    <abp-modal>
        <abp-modal-header title="Create New Product"></abp-modal-header>
        <abp-modal-body>
            <abp-input asp-for="Product.Name"/>
            <abp-input asp-for="Product.Description"/>
            <abp-input asp-for="Product.ReleaseDate"/>
        </abp-modal-body>
        <abp-modal-footer buttons="@AbpModalButtons.Save | @AbpModalButtons.Cancel"></abp-modal-footer>
    </abp-modal>
</form>
```
```

\* The `abp-modal` has been wrapped by the `form`. This is needed to place the `Save` and the `Cancel` buttons into the form. In this way, the `Save` button acts as the `submit` button for the `form`.

\* Used the [abp-input tag helpers](Tag-Helpers/Form-Elements.md) to simplify to create the form elements. Otherwise, you need to write more HTML.

#### \*\*ProductCreateModal.cshtml.cs Content:\*\*

```
```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
```

```

using Volo.Abp.AspNetCore.Mvc.UI.RazorPages;

namespace MyProject.Web.Pages.Products
{
    public class ProductCreateModalModel : AbpPageModel
    {
        [BindProperty]
        public PoductCreationDto Product { get; set; }

        public async Task OnGetAsync()
        {
            //TODO: Get logic, if available
        }

        public async Task<IActionResult> OnPostAsync()
        {
            //TODO: Save the Product...

            return NoContent();
        }
    }
}
```

```

- \* This is a simple `PageModal` class. The `[BindProperty]` make the form binding to the model when you post (submit) the form; The standard ASP.NET Core system.
- \* `OnPostAsync` returns `NoContent` (this method is defined by the base `AbpPageModel` class). Because we don't need to a return value in the client side, after the form post operation.

#### **\*\*PoductCreationDto:\*\***

`ProductCreateModalModel` uses a `PoductCreationDto` class defined as shown below:

```

```csharp
using System;
using System.ComponentModel.DataAnnotations;
using Volo.Abp.AspNetCore.Mvc.UI.Bootstrap.TagHelpers.Form;

namespace MyProject.Web.Pages.Products
{
    public class PoductCreationDto
    {
        [Required]
        [StringLength(128)]
        public string Name { get; set; }

        [TextArea(Rows = 4)]
        [StringLength(2000)]
        public string Description { get; set; }

        [DataType(DataType.Date)]
    }
}
```

```

```
 public DateTime ReleaseDate { get; set; }
 }
}...
`
```

\* `abp-input` Tag Helper can understand the data annotation attributes and uses them to shape and validate the form elements. See the [abp-input tag helpers](Tag-Helpers/Form-Elements.md) document to learn more.

#### #### Defining the Modal Manager

Again, create an `abp.ModalManager` object by setting the `viewUrl`, in the JavaScript file of the page that will use the modal:

```
```js  
var productCreateModal = new abp.ModalManager({  
    viewUrl: '/Products/ProductCreateModal'  
});  
```
```

#### #### Opening the Modal

Then open the modal whenever you need:

```
```js  
productCreateModal.open();  
```
```

You typically want to open the modal when something happens; For example, when the user clicks a button:

```
```js  
$('#OpenProductCreateModal').click(function() {  
    productCreateModal.open();  
});  
```
```

So, the complete code will be something like that (assuming you have a `button` with `id` is `OpenProductCreateModal` on the view side):

```
```js  
$(function () {  
  
    var productCreateModal = new abp.ModalManager({  
        viewUrl: '/Products/ProductCreateModal'  
    });  
  
    $('#OpenProductCreateModal').click(function () {  
        productCreateModal.open();  
    });  
  
});  
```
```

```
```
```

The resulting modal will be like that:

```
![modal-example-product-create](../../images/modal-example-product-create.png)
```

Saving the Modal

When you click to the `Save` button, the form is posted to the server. If the server returns a ****success response****, then the `onResult` event is triggered with some arguments including the server response and the modal is automatically closed.

An example callback that logs the arguments passed to the `onResult` method:

```
```js
productCreateModal.onResult(function() {
 console.log(arguments);
});```
````
```

If the server returns a failed response, it shows the error message returned from the server and keeps the modal open.

➤ See the **Modal Manager Reference** section below for other modal events.

Canceling the Modal

If you click to the Cancel button with some changes made but not saved, you get such a warning message:

```
![modal-manager-cancel-warning](../../images/modal-manager-cancel-warning.png)
```

If you don't want such a check & message, you can add `data-check-form-on-close="false"` attribute to your `form` element. Example:

```
```html
<form method="post"
 action="@Url.Page("/Products/ProductCreateModal")"
 data-check-form-on-close="false">
````
```

Form Validation

`ModalManager` automatically triggers the form validation when you click to the `Save` button or hit the `Enter` key on the form:

```
![modal-manager-validation](../../images/modal-manager-validation.png)
```

See the [\[Forms & Validation document\]](#) (`Forms-Validation.md`) to learn more about the validation.

Modals with Script Files

You may need to perform some logic for your modal. To do that, create a JavaScript file like below:

```
```js
abp.modals.ProductInfo = function () {

 function initModal(modalManager, args) {
 var $modal = modalManager.getModal();
 var $form = modalManager.getForm();

 $modal.find('h3').css('color', 'red');

 console.log('initialized the modal...');

 }

 return {
 initModal: initModal
 };
}
````
```

* This code simply adds a `ProductInfo` class into the `abp.modals` namespace. The `ProductInfo` class exposes a single public function: `initModal`.
* `initModal` method is called by the `ModalManager` once the modal HTML is inserted to DOM and ready for the initialization logic.
* `modalManager` parameter is the `ModalManager` object related to this modal instance. So, you can use any function on it in your code. See the **ModalManager Reference** section.

Then include this file to the page that you use the modal:

```
```html
<abp-script src="/Pages/Products/ProductInfoModal.js"/>
<abp-script src="/Pages/Products/Index.js"/>
````
```

* We've use the `abp-script` Tag Helper here. See the [Bundling & Minification](Bundling-Minification.md) document if you want to understand it. You can use the standard `script` tag. It doesn't matter for this case.

Finally, set the `modalClass` option while creating the `ModalManager` instance:

```
```js
var productInfoModal = new abp.ModalManager({
 viewUrl: '/Products/ProductInfoModal',
 modalClass: 'ProductInfo' //Matches to the abp.modals.ProductInfo
});
````
```

Lazy Loading the Script File

Instead of adding the `ProductInfoModal.js` to the page you use the modal, you can configure it to lazy load the script file when the first time the modal is opened.

Example:

```
```js
var productInfoModal = new abp.ModalManager({
 viewUrl: '/Products/ProductInfoModal',
 scriptUrl: '/Pages/Products/ProductInfoModal.js', //Lazy Load URL
 modalClass: 'ProductInfo'
});
```

\* `scriptUrl` is used to set the URL to load the script file of the modal.

\* In this case, you no longer need to include the `ProductInfoModal.js` to the page. It will be loaded on demand.

#### #### Tip: Bundling & Minification

While lazy loading seems cool at the beginning, it requires an additional call to the server when you first open the modal.

Instead, you can use the [Bundling & Minification](Bundling-Minification.md) system to create a bundle (that is a single and minified file on production) for all the used script files for a page:

```
```html
<abp-script-bundle>
    <abp-script src="/Pages/Products/ProductInfoModal.js"/>
    <abp-script src="/Pages/Products/Index.js"/>
</abp-script-bundle>
```

This is efficient if the script file is not large and frequently opened while users use the page.

Alternatively, you can define the `abp.modals.ProductInfo` class in the page's main JavaScript file if the modal is only and always used in the same page. In this case, you don't need to another external script file at all.

ModalManager Reference

Options

Options can be passed when you create a new `ModalManager` object:

```
```js
var productInfoModal = new abp.ModalManager({
 viewUrl: '/Products/ProductInfoModal',
 //...other options
});
```

Here, the list of all available options;

- \* `viewUrl` (required, `string`): The URL to lazy load the HTML of the modal.
- \* `scriptUrl` (optional, `string`): A URL to lazy load a JavaScript file. It is loaded only once, when the modal first opened.
- \* `modalClass` (optional, `string`): A JavaScript class defined in the `abp.modals` namespace that can be used to execute code related to the modal.
- \* `focusElement` (optional, `function or string`): Specifies the element that gets focus.

#### ### Functions

When you create a new `ModalManager` object, you can use its functions to perform operations on the modal. Example:

```
```js
var myModal = new abp.ModalManager({
    //...options
});

//Open the modal
myModal.open();

//Close the modal
myModal.close();
```
```

Here, the list of all available functions of the `ModalManager` object;

- \* `open([args])`: Opens the modal dialog. It can get an `args` object that is converted to query string while getting the `viewUrl` from the server. For example, if `args` is `{ productId: 42 }`, then the `ModalManager` passes `?productId=42` to the end of the `viewUrl` while loading the view from the server.
- \* `reopen()`: Opens the modal with the latest provided `args` for the `open()` method. So, it is a shortcut if you want to re-open the modal with the same `args`.
- \* `close()`: Closes the modal. The modal HTML is automatically removed from DOM once it has been closed.
- \* `getModalId()`: Gets the `id` attribute of the container that contains the view returned from the server. This is a unique id per modal and it doesn't change after you create the `ModalManager`.
- \* `getModal()`: Returns the modal wrapper DOM element (the HTML element with the `modal` CSS class) as a JQuery selection, so you can perform any JQuery method on it.
- \* `getForm()`: Returns the `form` HTML element as a JQuery selection, so you can perform any JQuery method on it. It returns `null` if the modal has no form inside it.
- \* `getArgs()`: Gets the latest arguments object provided while opening the modal.
- \* `getOptions()`: Gets the options object passed to the `ModalManager` constructor.
- \* `setResult(...)`: Triggers the `onResult` event with the provided arguments. You can pass zero or more arguments those are directly passed to the `onResult` event. This function is generally called by the modal script to notify the page that uses the modal.

#### ### Events

When you create a new `ModalManager` object, you can use its functions register to events of the modal. Examples:

```
```js
var myModal = new abp.ModalManager({
    //...options
});

myModal.onOpen(function () {
    console.log('opened the modal...');
});

myModal.onClose(function () {
    console.log('closed the modal...');
});
```

```

Here, the list of all available functions to register to events of the `ModalManager` object;

- \* `onOpen(callback)` : Registers a callback function to get notified once the modal is opened. It is triggered when the modal is completely visible on the UI.
- \* `onClose(callback)` : Registers a callback function to get notified once the modal is closed. It is triggered when the modal is completely invisible on the UI.
- \* `onResult(callback)` : Registers a callback function that is triggered when the `setResult(...)` method is called. All the parameters sent to the `setResult` method is passed to the callback.

### 10.1.5 Data Tables

#### # ASP.NET Core MVC / Razor Pages: Data Tables

A Data Table (aka Data Grid) is a UI component to show tabular data to the users. There are a lot of Data table components/libraries and **\*\*you can use any one you like\*\*** with the ABP Framework. However, the startup templates come with the [\[DataTables.Net\]\(https://datatables.net/\)](#) library as **\*\*pre-installed and configured\*\***. ABP Framework provides adapters for this library and make it easy to use with the API endpoints.

An example screenshot from the user management page that shows the user list in a data table:

![datatables-example](../../images/datatables-example.png)

#### ## DataTables.Net Integration

First of all, you can follow the official documentation to understand how the [\[DataTables.Net\]\(https://datatables.net/\)](#) works. This section will focus on the ABP addons & integration points rather than fully covering the usage of this library.

#### ### A Quick Example

You can follow the [web application development tutorial](<https://docs.abp.io/en/abp/latest/Tutorials/Part-1?UI=MVC>) for a complete example application that uses the DataTables.Net as the Data Table. This section shows a minimalist example.

You do nothing to add DataTables.Net library to the page since it is already added to the global [bundle](Bundling-Minification.md) by default.

First, add an `abp-table` as shown below, with an `id`:

```
```html
<abp-table striped-rows="true" id="BooksTable"></abp-table>
````
```

> `abp-table` is a [Tag Helper](Tag-Helpers/Index.md) defined by the ABP Framework, but a simple `<table...>` tag would also work.

Then call the `DataTable` plugin on the table selector:

```
```js
var dataTable = $('#BooksTable').DataTable(
    abp.libs.datatables.normalizeConfiguration({
        serverSide: true,
        paging: true,
        order: [[1, "asc"]],
        searching: false,
        ajax: abp.libs.datatables.createAjax(acme.bookStore.books.book.getList),
        columnDefs: [
            {
                title: l('Actions'),
                rowAction: {
                    items:
                    [
                        {
                            text: l('Edit'),
                            action: function (data) {
                                //...
                            }
                        }
                    ]
                }
            },
            {
                title: l('Name'),
                data: "name"
            },
            {
                title: l('PublishDate'),
                data: "publishDate",
                render: function (data) {

```

```

        return luxon
            .DateTime
            .fromISO(data, {
                locale: abp.localization.currentCulture.name
            }).toLocaleString();
        }
    },
{
    title: l('Price'),
    data: "price"
}
]
})
);
```

```

The example code above uses some ABP integration features those will be explained in the next sections.

#### ### Configuration Normalization

``abp.lib.datatables.normalizeConfiguration`` function takes a DataTables configuration and normalizes to simplify it;

- \* Sets `scrollX` option to `true`, if not set.
- \* Sets `target` index for the column definitions.
- \* Sets the `language` option to [localize](../../Localization.md) the table in the current language.

#### #### Default Configuration

``normalizeConfiguration`` uses the default configuration. You can change the default configuration using the ``abp.lib.datatables.defaultConfigurations`` object. Example:

```

```js
abp.lib.datatables.defaultConfigurations.scrollX = false;
```

```

Here, the all configuration options;

- \* `scrollX`: `false` by default.
- \* `dom`: Default value is `<"dataTable\_filters" f>rt<"row dataTable\_footer"><"col-auto" i><"col-auto" p>>`.
- \* `language`: A function that returns the localization text using the current language.

#### ### AJAX Adapter

DataTables.Net has its own expected data format while getting results of an AJAX call to the server to get the table data. They are especially related how paging and sorting parameters are sent and received. ABP Framework also offers its own conventions for the client-server [AJAX](JavaScript-API/Ajax.md) communication.

The ``abp.lib.datatables.createAjax`` method (used in the example above) adapts request and response data format and perfectly works with the [Dynamic JavaScript Client Proxy] (Dynamic-JavaScript-Proxies.md) system.

This works automatically, so most of the times you don't need to know how it works. See the [DTO document] (../../Data-Transfer-Objects.md) if you want to learn more about ``IPagedAndSortedResultRequest``, ``IPagedResult`` and other standard interfaces and base DTO classes those are used in client to server communication.

The ``createAjax`` also supports you to customize request parameters and handle the responses.

**\*\*Example:\*\***

```
```csharp
var inputAction = function (requestData, dataTableSettings) {
    return {
        id: $('#Id').val(),
        name: $('#Name').val(),
    };
};

var responseCallback = function(result) {
    // your custom code.

    return {
        recordsTotal: result.totalCount,
        recordsFiltered: result.totalCount,
        data: result.items
    };
};

ajax: abp.lib.datatables.createAjax(acme.bookStore.books.book.getList, inputAction,
responseCallback)
```
```

If you don't need access or modify the ``requestData`` or the ``dataTableSettings``, you can specify a simple object as the second parameter.

```
```js
ajax: abp.lib.datatables.createAjax(
    acme.bookStore.books.book.getList,
    { id: $('#Id').val(), name: $('#Name').val() }
)
```
```

#### ### Row Actions

``rowAction`` is an option defined by the ABP Framework to the column definitions to show a drop down button to take actions for a row in the table.

The example screenshot below shows the actions for each user in the user management table:

```
![datatables-example](../../images/datatables-row-actions.png)
```

`rowAction` is defined as a part of a column definition:

```
```csharp
{
    title: l('Actions'),
    rowAction: {
        //TODO: CONFIGURATION
    }
},```

```

****Example: Show *Edit* and *Delete* actions for a book row****

```
```js
{
 title: l('Actions'),
 rowAction: {
 items: [
 {
 text: l('Edit'),
 action: function (data) {
 //TODO: Open a modal to edit the book
 }
 },
 {
 text: l('Delete'),
 confirmMessage: function (data) {
 return "Are you sure to delete the book " + data.record.name;
 },
 action: function (data) {
 acme.bookStore.books.book
 .delete(data.record.id)
 .then(function () {
 abp.notify.info("Successfully deleted!");
 data.table.ajax.reload();
 });
 }
 }
]
 }
},```

```

#### Action Items

`items` is an array of action definitions. An action definition can have the following options;

- \* `text`: The text (a `string`) for this action to be shown in the actions drop down.
- \* `action`: A `function` that is executed when the user clicks to the action. The function takes a `data` argument that has the following fields;
  - \* `data.record`: This is the data object related to the row. You can access the data fields like `data.record.id`, `data.record.name` ... etc.
  - \* `data.table`: The DataTables instance.
- \* `confirmMessage`: A `function` (see the example above) that returns a message (`string`) to show a dialog to get a confirmation from the user before executing the `action`.

Example confirmation dialog:

```
![datatables-row-actions-confirmation](../../images/datatables-row-actions-confirmation.png)
```

You can use the [localization] (JavaScript-API/Localization.md) system to show a localized message.

- \* `visible`: A `bool` or a `function` that returns a `bool`. If the result is `false`, then the action is not shown in the actions dropdown. This is generally combined by the [authorization] (JavaScript-API/Auth.md) system to hide the action if the user has no permission to take this action. Example:

```
```js
visible: abp.auth.isGranted('BookStore.Books.Delete');
````
```

If you define a `function`, then the `function` has two arguments: `record` (the data object of the related row) and the `table` (the DataTable instance). So, you can decide to show/hide the action dynamically, based on the row data and other conditions.

- \* `iconClass`: Can be used to show a font-icon, like a [Font Awesome] (<https://fontawesome.com/>) icon (ex: `fas fa-trash-alt`), near to the action text. Example screenshot:

```
![datatables-row-actions-icon](../../images/datatables-row-actions-icon.png)
```

- \* `enabled`: A `function` that returns a `bool` to disable the action. The `function` takes a `data` object with two fields: `data.record` is the data object related to the row and `data.table` is the DataTables instance.
- \* `displayNameHtml`: Set this to `true` is the `text` value contains HTML tags.

There are some rules with the action items;

- \* If none of the action items is visible then the actions column is not rendered.

### ### Data Format

#### #### The Problem

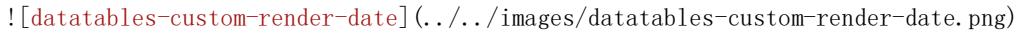
See the \*Creation Time\* column in the example below:

```
```js
```

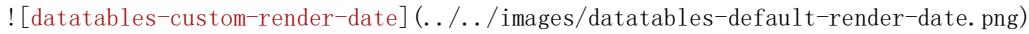
```

{
    title: l('CreationTime'),
    data: "creationTime",
    render: function (data) {
        return luxon
            .DateTime
            .fromISO(data, {
                locale: abp.localization.currentCulture.name
            }).toLocaleString(luxon.DateTime.DATETIME_SHORT);
    }
}
````
```

The `render` is a standard DataTables option to render the column content by a custom function. This example uses the [luxon](<https://moment.github.io/luxon/>) library (which is installed by default) to write a human readable value of the `creationTime` in the current user's language. Example output of the column:

 ![[datatables-custom-render-date]](../../images/datatables-custom-render-date.png)

If you don't define the render option, then the result will be ugly and not user friendly:

 ![[datatables-custom-render-date]](../../images/datatables-default-render-date.png)

However, rendering a `DateTime` is almost same and repeating the same rendering logic everywhere is against to the DRY (Don't Repeat Yourself!) principle.

#### #### dataFormat Option

`dataFormat` column option specifies the data format that is used to render the column data. The same output could be accomplished using the following column definition:

```

````js
{
    title: l('CreationTime'),
    data: "creationTime",
    dataFormat: 'datetime'
}
````
```

`dataFormat: 'datetime'` specifies the data format for this column. There are a few pre-defined `dataFormat`s:

- \* `boolean`: Shows a `check` icon for `true` and `times` icon for `false` value and useful to render `bool` values.
- \* `date`: Shows date part of a `DateTime` value, formatted based on the current culture.
- \* `datetime`: Shows date & time (excluding seconds) of a `DateTime` value, formatted based on the current culture.

#### ### Default Renderers

``abp.lib.datatables.defaultRenderers`` option allows you to define new data formats and set renderers for them.

#### \*\*Example: Render male / female icons based on the gender\*\*

```
```js
abp.lib.datatables.defaultRenderers['gender'] = function(value) {
    if (value === 'f') {
        return '<i class="fa fa-venus"></i>';
    } else {
        return '<i class="fa fa-mars"></i>';
    }
};```
```

```

Assuming that the possible values for a column data is `f` and `m`, the `gender` data format shows female/male icons instead of `f` and `m` texts. You can now set `dataFormat: 'gender'` for a column definition that has the proper data values.

➤ You can write the default renderers in a single JavaScript file and add it to the [Global Script Bundle] (Bundling-Minification.md), so you can reuse them in all the pages.

## ## Other Data Grids

You can use any library you like. For example, [see this article](https://community.abp.io/articles/using-devextreme-components-with-the-abp-framework-zb8z7yqv) to learn how to use DevExtreme Data Grid in your applications.

### 10.1.6 Auto-Complete Select

#### # ASP.NET Core MVC / Razor Pages: Auto-Complete Select

A simple select component sometimes isn't useful with a huge amount of data. ABP Provides a select implementation that works with pagination and server-side search via using [Select2](https://select2.org/). It works with single or multiple choices well.

A screenshot can be shown below.

```
| Single | Multiple |
| --- | --- |
| ! [autocomplete-select-example] (../../images/abp-select2-single.png) | ! [autocomplete-select-example] (../../images/abp-select2-multiple.png) |
```

## ## Getting Started

This is a core feature and it's used by the ABP Framework. There is no custom installation or additional packages required.

## ## Usage

A simple usage is presented below.

```

```html
<select asp-for="Book.AuthorId"
    class="auto-complete-select"
    data-autocomplete-api-url="/api/app/author"
    data-autocomplete-display-property="name"
    data-autocomplete-value-property="id"
    data-autocomplete-items-property="items"
    data-autocomplete-filter-param-name="filter"
    data-autocomplete-allow-clear="true">

    <!-- You can define selected option(s) here -->
    <option selected value="@SelectedAuthor.Id">@SelectedAuthor.Name</option>
</select>
```

```

The select must have the `auto-complete-select` class and the following attributes:

- `data-autocomplete-api-url`: \* API Endpoint url to get select items. \*\*GET\*\* request will be sent to this url.
- `data-autocomplete-display-property`: \* Property name to display. *(For example: `name` or `title`. Property name of entity/dto.)*.
- `data-autocomplete-value-property`: \* Identifier property name. *(For example: `id`)*.
- `data-autocomplete-items-property`: \* Property name of collection in response object. *(For example: `items`)*.
- `data-autocomplete-filter-param-name`: \* Filter text property name. *(For example: `filter`)*.
- `data-autocomplete-selected-item-name`: Text to display as selected item.
- `data-autocomplete-parent-selector`: jQuery selector expression for parent DOM. *(If it's in a modal, it's suggested to send the modal selector as this parameter)*.
- `data-autocomplete-allow-clear`: If `true`, it'll allow to clear the selected value. Default value: `false`.
- `data-autocomplete-placeholder`: Placeholder text to display when no value is selected.

Also, selected value(s) should be defined with the `<option>` tags inside select, since pagination is applied and the selected options might haven't loaded yet.

### ### Multiple Choices

AutoComplete Select supports multiple choices. If the select tag has a `multiple` attribute, it'll allow to choose multiple options.

```

```html
<select asp-for="Book.TagsIds"
    class="auto-complete-select"
    multiple="multiple"
    data-autocomplete-api-url="/api/app/tags"
    data-autocomplete-display-property="name"
    data-autocomplete-value-property="id"
    data-autocomplete-items-property="items"
    data-autocomplete-filter-param-name="filter">
    @foreach(var tag in SelectedTags)

```

```

    {
        <option selected value="@tag.Id">@tag.Name</option>
    }
</select>
```

```

It'll be automatically bound to a collection of defined value type.

```

```csharp
    public List<Guid> TagIds { get; set; }
```

```

## ## Notices

If the authenticated user doesn't have permission on the given URL, the user will get an authorization error. Be careful while designing this kind of UIs.

You can create a specific, [unauthorized](../../Authorization.md) endpoint/method to get the list of items, so the page can retrieve lookup data of dependent entity without giving the entire read permission to users.

### 10.1.7 Page Alerts

#### # ASP.NET Core MVC / Razor Pages: Page Alerts

It is common to show error, warning or information alerts to inform the user. An example *\*Service Interruption\** alert is shown below:

```
![page-alert-example](../../images/page-alert-example.png)
```

## ## Basic Usage

If you directly or indirectly inherit from `AbpPageModel`, you can use the `Alerts` property to add alerts to be rendered after the request completes.

### **\*\*Example: Show a Warning alert\*\***

```

```csharp
namespace MyProject.Web.Pages
{
    public class IndexModel : MyProjectPageModel //or inherit from AbpPageModel
    {
        public void OnGet()
        {
            Alerts.Warning(
                text: "We will have a service interruption between 02:00 AM and 04:00 AM
at October 23, 2023!",
                title: "Service Interruption"
            );
        }
    }
}
```

```

This usage renders an alert that was shown above. If you need to localize the messages, you can always use the standard [localization](../../Localization.md) system.

#### ### Exceptions / Invalid Model States

It is typical to show alerts when you manually handle exceptions (with try/catch statements) or want to handle `!ModelState.IsValid` case and warn the user. For example, the Account Module shows a warning if user enters an incorrect username or password:

```
![page-alert-account-layout](../../images/page-alert-account-layout.png)
```

➤ Note that you generally don't need to manually handle exceptions since ABP Framework provides an automatic [exception handling](../../Exception-Handling.md) system.

#### ### Alert Types

`Warning` is used to show a warning alert. Other common methods are `Info`, `Danger` and `Success`.

Beside the standard methods, you can use the `Alerts.Add` method by passing an `AlertType` enum with one of these values: `Default`, `Primary`, `Secondary`, `Success`, `Danger`, `Warning`, `Info`, `Light`, `Dark`.

#### ### Dismissible

All alert methods gets an optional `-dismissible` parameter. Default value is `true` which makes the alert box dismissible. Set it to `false` to create a sticky alert box.

#### ## IAlertManager

If you need to add alert messages from another part of your code, you can inject the `IAlertManager` service and use its `Alerts` list.

##### \*\*Example: Inject the `IAlertManager`\*\*

```
```csharp
using Volo.Abp.AspNetCore.Mvc.UI.Alerts;
using Volo.Abp.DependencyInjection;

namespace MyProject.Web.Pages
{
    public class MyService : ITransientDependency
    {
        private readonly IAlertManager _alertManager;

        public MyService(IAlertManager alertManager)
        {
            _alertManager = alertManager;
        }

        public void Test()
        {
```

```
        _alertManager.Alerts.Add(AlertType.Danger, "Test message!");
    }
}
```

```

## Notes

### ### AJAX Requests

Page Alert system was designed to be used in a regular full page request. It is not for AJAX/partial requests. The alerts are rendered in the page layout, so a full page refresh is needed.

For AJAX requests, it is more proper to throw exceptions (e.g. `UserFriendlyException`). See the [exception handling](../../Exception-Handling.md) document.

#### 10.1.8 Dynamic JavaScript API Client Proxies

##### # Dynamic JavaScript API Client Proxies

It is typical to consume your HTTP APIs from your JavaScript code. To do that, you normally deal with low level AJAX calls, like `$.ajax`, or better [abp.ajax](JavaScript-API/Ajax.md). ABP Framework provides **\*\*a better way\*\*** to call your HTTP APIs from your JavaScript code: JavaScript API Client Proxies!

##### ## Static vs Dynamic JavaScript Client Proxies

ABP provides **\*\*two types\*\*** of client proxy generation system. This document explains the **\*\*dynamic client proxies\*\***, which generates client-side proxies on runtime. You can also see the [Static JavaScript API Client Proxies](Static-JavaScript-Proxies.md) documentation to learn how to generate proxies on development time.

Development-time (static) client proxy generation has a **\*\*slight performance advantage\*\*** since it doesn't need to obtain the HTTP API definition on runtime. However, you should **\*\*re-generate\*\*** the client proxy code whenever you change your API endpoint definition. On the other hand, dynamic client proxies are generated on runtime and provides an **\*\*easier development experience\*\***.

##### ## A Quick Example

Assume that you have an application service defined as shown below:

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;

namespace Acme.BookStore.Authors
{
```

```

public interface IAuthorAppService : IApplicationService
{
    Task<AuthorDto> GetAsync(Guid id);

    Task<PagedResultDto<AuthorDto>> GetListAsync(GetAuthorListDto input);

    Task<AuthorDto> CreateAsync(CreateAuthorDto input);

    Task UpdateAsync(Guid id, UpdateAuthorDto input);

    Task DeleteAsync(Guid id);
}
```

```

> You can follow the [web application development tutorial](../../Tutorials/Part-1.md) to learn how to create [application services](../../Application-Services.md), expose them as [HTTP APIs](../../API/Auto-API-Controllers.md) and consume from the JavaScript code as a complete example.

You can call any of the methods just like calling a JavaScript function. The JavaScript function has the identical function **\*\*name\*\***, **\*\*parameters\*\*** and the **\*\*return value\*\*** with the C# method.

#### **\*\*Example: Get the authors list\*\***

```

```js
acme.bookStore.authors.author.getList({
    maxResultCount: 10
}).then(function(result) {
    console.log(result.items);
});
```

```

#### **\*\*Example: Delete an author\*\***

```

```js
acme.bookStore.authors.author
    .delete('7245a066-5457-4941-8aa7-3004778775f0') //Get id from somewhere!
    .then(function() {
        abp.notify.info('Successfully deleted!');
    });
```

```

#### **## AJAX Details**

JavaScript client proxy functions use the [abp.ajax](JavaScript-API/Ajax.md) under the hood. So, you have the same benefits like **\*\*automatic error handling\*\***. Also, you can fully control the AJAX call by providing the options.

#### **### The Return Value**

Every function returns a [Deferred object] (<https://api.jquery.com/category/deferred-object/>). That means you can chain with `then` to get the result, `catch` to handle the error, `always` to perform an action once the operation completes (success or failed).

### ### AJAX Options

Every function gets an additional **\*\*last parameter\*\*** after your own parameters. The last parameter is called as `ajaxParams`. It is an object that overrides the AJAX options.

**\*\*Example: Set `type` and `dataType` AJAX options\*\***

```
```js
acme.bookStore.authors.author
  .delete('7245a066-5457-4941-8aa7-3004778775f0', {
    type: 'POST',
    dataType: 'xml'
  })
  .then(function() {
    abp.notify.info('Successfully deleted!');
  });
````
```

See the [jQuery.ajax] (<https://api.jquery.com/jQuery.ajax/>) documentation for all the available options.

### ## Service Proxy Script Endpoint

The magic is done by the `/Abp/ServiceProxyScript` endpoint defined by the ABP Framework and automatically added to the layout. You can visit this endpoint in your application to see the client proxy function definitions. This script file is automatically generated by the ABP Framework based on the server side method definitions and the related HTTP endpoint details.

### ## See Also

- \* [Static JavaScript API Client Proxies] ([Static-JavaScript-Proxies.md](#))
- \* [Auto API Controllers] ([..../API/Auto-API-Controllers.md](#))
- \* [Web Application Development Tutorial] ([..../Tutorials/Part-1.md](#))

### 10.1.9 Static JavaScript API Client Proxies

#### # Static JavaScript API Client Proxies

It is typical to consume your HTTP APIs from your JavaScript code. To do that, you normally deal with low level AJAX calls, like `$.ajax`, or better [abp.ajax] ([JavaScript-API/Ajax.md](#)). ABP Framework provides **\*\*a better way\*\*** to call your HTTP APIs from your JavaScript code: JavaScript API Client Proxies!

#### ## Static vs Dynamic JavaScript Client Proxies

ABP provides **\*\*two types\*\*** of client proxy generation system. This document explains the **\*\*static client proxies\*\***, which generates client-side code in your development time. You can also see the [\[Dynamic JavaScript API Client Proxies\]](#) (`Dynamic-JavaScript-Proxies.md`) documentation to learn how to use proxies generated on runtime.

Development-time (static) client proxy generation has a **\*\*slight performance advantage\*\*** since it doesn't need to obtain the HTTP API definition on runtime. However, you should **\*\*re-generate\*\*** the client proxy code whenever you change your API endpoint definition. On the other hand, dynamic client proxies are generated on runtime and provides an **\*\*easier development experience\*\***.

## ## A Quick Example

### ### The Application Service

Assume that you have an application service defined as shown below:

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.Application.Dtos;
using Volo.Abp.Application.Services;

namespace Acme.BookStore.Authors
{
    public interface IAuthorAppService : IApplicationService
    {
        Task<AuthorDto> GetAsync(Guid id);

        Task<PagedResultDto<AuthorDto>> GetListAsync(GetAuthorListDto input);

        Task<AuthorDto> CreateAsync(CreateAuthorDto input);

        Task UpdateAsync(Guid id, UpdateAuthorDto input);

        Task DeleteAsync(Guid id);
    }
}
````
```

> You can follow the [\[web application development tutorial\]](#) (`../../Tutorials/Part-1.md`) to learn how to create [\[application services\]](#) (`../../Application-Services.md`), expose them as [\[HTTP APIs\]](#) (`../../API/Auto-API-Controllers.md`) and consume from the JavaScript code as a complete example.

### ### Generating the JavaScript Code

Server side must be up and running while generating the client proxy code. So, first run the application that hosts your HTTP APIs (can be the Web application or the `HttpApi.Host` application depending on your solution structure).

Open a command-line terminal in the root folder of your web project (`.\csproj`) and type the following command:

```
```bash
abp generate-proxy -t js -u https://localhost:53929/
````
```

> If you haven't installed yet, you should install the [ABP CLI](./CLI.md). Change the example URL to your application's root URL.

This command should generate the following files under the `'ClientProxies'` folder:

```
![static-js-proxy-example](../../images/static-js-proxy-example.png)
```

`'app-proxy.js'` is the generated proxy file in this example. Here, an example proxy function in this file:

```
```js
acme.bookStore.authors.author.get = function(id, ajaxParams) {
    return abp.ajax($.extend(true, {
        url: abp.appPath + 'api/app/author/' + id + '',
        type: 'GET'
    }, ajaxParams));
}
````
```

> `'generate-proxy'` command generates proxies for only the APIs you've defined in your application (assumes `'app'` as the module name). If you are developing a modular application, you can specify the `'-m'` (or `--module`) parameter to specify the module you want to generate proxies. See the `*generate-proxy*` section in the [ABP CLI](./CLI.md) documentation for other options.

### ### Using the Proxy Functions

To use the proxy functions, first import the `'app-proxy.js'` file into your page:

```
```html
<abp-script src="/client-proxies/app-proxy.js"/>
````
```

> We've used the [abp-script tag helper](Bundling-Minification.md) in this example. You could use the standard `'script'` tag, but the `'abp-script'` is the recommended way of importing JavaScript files to your pages.

Now, you can call any of the application service methods from your JavaScript code, just like calling a JavaScript function. The JavaScript function has the identical function **\*\*name\*\***, **\*\*parameters\*\*** and the **\*\*return value\*\*** with the C# method.

#### **\*\*Example: Get a single author\*\***

```
```js
acme.bookStore.authors.author
```

```
    .get("7245a066-5457-4941-8aa7-3004778775f0") //Get id from somewhere!
    .then(function(result) {
        console.log(result);
    });
````
```

#### \*\*Example: Get the authors list\*\*

```
```js
acme.bookStore.authors.author.getList({
    maxResultCount: 10
}).then(function(result) {
    console.log(result.items);
});
````
```

#### \*\*Example: Delete an author\*\*

```
```js
acme.bookStore.authors.author
    .delete('7245a066-5457-4941-8aa7-3004778775f0') //Get id from somewhere!
    .then(function() {
        abp.notify.info('Successfully deleted!');
    });
````
```

#### ## Disabling Dynamic JavaScript Proxies

When you create an application or module, the [dynamic client proxy generation] (Dynamic-JavaScript-Proxies.md) approach is used by default. If you want to use the statically generated client proxies for your application, you should explicitly disable it for your application or module in the `ConfigureServices` method of your [module class](../../../../Module-Development-Basics.md) as like in the following example:

```
```csharp
Configure<DynamicJavaScriptProxyOptions>(options =>
{
    options.DisableModule("app");
});
````
```

`app` represents the main application in this example, which works if you are creating an application. If you are developing an application module, then use your module's name.

#### ## AJAX Details

JavaScript client proxy functions use the [abp.ajax] (JavaScript-API/Ajax.md) under the hood. So, you have the same benefits like **\*\*automatic error handling\*\***. Also, you can fully control the AJAX call by providing the options.

#### ### The Return Value

Every function returns a [Deferred object] (<https://api.jquery.com/category/deferred-object/>). That means you can chain with `then` to get the result, `catch` to handle the error, `always` to perform an action once the operation completes (success or failed).

### ### AJAX Options

Every function gets an additional **\*\*last parameter\*\*** after your own parameters. The last parameter is called as `ajaxParams`. It is an object that overrides the AJAX options.

**\*\*Example: Set `type` and `dataType` AJAX options\*\***

```
```js
acme.bookStore.authors.author
.delete('7245a066-5457-4941-8aa7-3004778775f0', {
    type: 'POST',
    dataType: 'xml'
})
.then(function() {
    abp.notify.info('Successfully deleted!');
});
````
```

See the [jQuery.ajax] (<https://api.jquery.com/jQuery.ajax/>) documentation for all the available options.

### ## See Also

- \* [Dynamic JavaScript API Client Proxies] ([Dynamic-JavaScript-Proxies.md](#))
- \* [Auto API Controllers] ([./../API/Auto-API-Controllers.md](#))
- \* [Web Application Development Tutorial] ([./../Tutorials/Part-1.md](#))

## 10.1.10 Client Side Package Management

### ## ASP.NET Core MVC Client Side Package Management

ABP framework can work with any type of client side package management systems. You can even decide to use no package management system and manage your dependencies manually.

However, ABP framework works best with **\*\*NPM/Yarn\*\***. By default, built-in modules are configured to work with NPM/Yarn.

Finally, we suggest the [\*\*Yarn\*\*] (<https://classic.yarnpkg.com/>) over the NPM since it's faster, stable and also compatible with the NPM.

### ### @ABP NPM Packages

ABP is a modular platform. Every developer can create modules and the modules should work together in a **\*\*compatible\*\*** and **\*\*stable\*\*** state.

One challenge is the **versions of the dependant NPM packages**. What if two different modules use the same JavaScript library but its different (and potentially incompatible) versions.

To solve the versioning problem, we created a **standard set of packages** those depends on some common third-party libraries. Some example packages are [\[@abp/jquery\]](https://www.npmjs.com/package/@abp/jquery) (<https://www.npmjs.com/package/@abp/jquery>), [\[@abp/bootstrap\]](https://www.npmjs.com/package/@abp/bootstrap) (<https://www.npmjs.com/package/@abp/bootstrap>) and [\[@abp/font-awesome\]](https://www.npmjs.com/package/@abp/font-awesome) (<https://www.npmjs.com/package/@abp/font-awesome>). You can see the **list of packages** from the [\[GitHub repository\]](https://github.com/volosoft/abp/tree/master/npm/packs) (<https://github.com/volosoft/abp/tree/master/npm/packs>).

The benefit of a **standard package** is:

- \* It depends on a **standard version** of a package. Depending on this package is **safe** because all modules depend on the same version.
- \* It contains the mappings copy library resources (js, css, img... files) from the `node\_modules` folder to `wwwroot/libs` folder. See the *\*Mapping The Library Resources\** section for more.

Depending on a standard package is easy. Just add it to your **package.json** file like you normally do. Example:

```
```json
{
  ...
  "dependencies": {
    "@abp/bootstrap": "^1.0.0"
  }
}
```

```

It's suggested to depend on a standard package instead of directly depending on a third-party package.

#### #### Package Installation

After depending on a NPM package, all you should do is to run the **yarn** command from the command line to install all the packages and their dependencies:

```
```bash
yarn
```

```

Alternatively, you can use `npm install` but [\[Yarn\]](https://classic.yarnpkg.com/) (<https://classic.yarnpkg.com/>) is suggested as mentioned before.

#### #### Package Contribution

If you need a third-party NPM package that is not in the standard set of packages, you can create a Pull Request on the Github [\[repository\]](https://github.com/volosoft/abp) (<https://github.com/volosoft/abp>). A pull request that follows these rules is accepted:

- \* Package name should be named as `@abp/package-name` for a `package-name` on NPM (example: `@abp/bootstrap` for the `bootstrap` package).
- \* It should be the \*\*latest stable\*\* version of the package.
- \* It should only depend a \*\*single\*\* third-party package. It can depend on multiple `@abp/\*` packages.
- \* The package should include a `abp.resourcemapping.js` file formatted as defined in the *Mapping The Library Resources* section. This file should only map resources for the depended package.
- \* You also need to create [bundle contributor(s)](Bundling-Minification.md) for the package you have created.

See current standard packages for examples.

#### ### Mapping The Library Resources

Using NPM packages and NPM/Yarn tool is the de facto standard for client side libraries. NPM/Yarn tool creates a \*\*node\_modules\*\* folder in the root folder of your web project.

Next challenge is copying needed resources (js, css, img... files) from the `node\_modules` into a folder inside the \*\*wwwroot\*\* folder to make it accessible to the clients/browsers.

ABP CLI's `abp install-libs` command \*\*copies resources\*\* from \*\*node\_modules\*\* to \*\*wwwroot/libs\*\* folder. Each \*\*standard package\*\* (see the *@ABP NPM Packages* section) defines the mapping for its own files. So, most of the time, you only configure dependencies.

The \*\*startup templates\*\* are already configured to work all these out of the box. This section will explain the configuration options.

#### #### Resource Mapping Definition File

A module should define a JavaScript file named `abp.resourcemapping.js` which is formatted as in the example below:

```
```json
module.exports = {
  aliases: {
    "@node_modules": "./node_modules",
    "@libs": "./wwwroot/libs"
  },
  clean: [
    "@libs",
    "!@libs/**/foo.txt"
  ],
  mappings: {
  }
}
```

```

- \* **aliases** section defines standard aliases (placeholders) that can be used in the mapping paths. **@node\_modules** and **@libs** are required (by the standard packages), you can define your own aliases to reduce duplication.
- \* **clean** section is a list of folders to clean before copying the files. Glob matching and negation is enabled, so you can fine-tune what to delete and keep. The example above will clean everything inside `./wwwroot/libs`, but keep any `foo.txt` files.
- \* **mappings** section is a list of mappings of files/folders to copy. This example does not copy any resource itself, but depends on a standard package.

An example mapping configuration is shown below:

```
```json
mappings: {
  "@node_modules/bootstrap/dist/css/bootstrap.css": "@libs/bootstrap/css/",
  "@node_modules/bootstrap/dist/js/bootstrap.bundle.js": "@libs/bootstrap/js/",
  "@node_modules/bootstrap-datepicker/dist/locales/*.": "@libs/bootstrap-
datepicker/locales/",
  "@node_modules/bootstrap-v4-rtl/dist/**/*": "@libs/bootstrap-v4-rtl/dist/"
}
```

```

#### #### install-libs Command

Once you properly configure the `abp.resourcemapping.js` file, you can run the following ABP CLI command from the command line:

```
```bash
abp install-libs
```

```

When you run this command, all packages will copy their own resources into the `wwwroot/libs` folder. Running `abp install-libs` is only necessary if you make a change in your dependencies in the **package.json** file.

#### #### See Also

- \* [Bundling & Minification] (Bundling-Minification.md)
- \* [Theming] (Theming.md)

### 10.1.11 Bundling & Minification

#### # ASP.NET Core MVC Bundling & Minification

There are many ways of bundling & minification of client side resources (JavaScript and CSS files). Most common ways are:

- \* Using the [Bundler & Minifier] (<https://marketplace.visualstudio.com/items?itemName=MadsKristensen.BundlerMinifier>)

er) Visual Studio extension or the [NuGet package](<https://www.nuget.org/packages/BuildBundlerMinifier/>).  
\* Using [Gulp](<https://gulpjs.com/>)/[Grunt](<https://gruntjs.com/>) task managers and their plugins.

ABP offers a simple, dynamic, powerful, modular and built-in way.

## ## Volo.Abp.AspNetCore.Mvc.UI.Bundling Package

➤ This package is already installed by default with the startup templates. So, most of the time, you don't need to install it manually.

If you're not using a startup template, you can use the [ABP CLI]([..../CLI.md](#)) to install it to your project. Execute the following command in the folder that contains the .csproj file of your project:

```
~~~~
```

```
abp add-package Volo.Abp.AspNetCore.Mvc.UI.Bundling
```

```
~~~~
```

➤ If you haven't done it yet, you first need to install the [ABP CLI]([..../CLI.md](#)). For other installation options, see [the package description page](<https://abp.io/package-detail/Volo.Abp.AspNetCore.Mvc.UI.Bundling>).

## ## Razor Bundling Tag Helpers

The simplest way of creating a bundle is to use `abp-script-bundle` or `abp-style-bundle` tag helpers. Example:

```
~~~~html
<abp-style-bundle name="MyGlobalBundle">
    <abp-style src="/libs/bootstrap/css/bootstrap.css" />
    <abp-style src="/libs/font-awesome/css/font-awesome.css" />
    <abp-style src="/libs/toastr/toastr.css" />
    <abp-style src="/styles/my-global-style.css" />
</abp-style-bundle>
~~~~
```

This bundle defines a style bundle with a **unique name**: `MyGlobalBundle`. It's very easy to understand how to use it. Let's see how it *works*:

- \* ABP creates the bundle as **lazy** from the provided files when it's **first requested**. For the subsequent calls, it's returned from the **cache**. That means if you conditionally add the files to the bundle, it's executed only once and any changes of the condition will not effect the bundle for the next requests.
- \* ABP adds bundle files **individually** to the page for the `development` environment. It automatically bundles & minifies for other environments (`staging`, `production`...). See the *Bundling Mode* section to change that behavior.
- \* The bundle files may be **physical** files or **virtual/embedded** files]([..../Virtual-File-System.md](#)).
- \* ABP automatically adds **version query string** to the bundle file URL to prevent browsers from caching when the bundle is being updated. (like ?\_v=67872834243042 -

generated from last change date of the related files). The versioning works even if the bundle files are individually added to the page (on the development environment).

### ### Importing The Bundling Tag Helpers

> This is already imported by default with the startup templates. So, most of the time, you don't need to add it manually.

In order to use bundle tag helpers, you need to add it into your `\\_ViewImports.cshtml` file or into your page:

```
````  
@addTagHelper *, Volo.Abp.AspNetCore.Mvc.UI.Bundling  
``` `
```

### ### Unnamed Bundles

The `name` is **optional** for the razor bundle tag helpers. If you don't define a name, it's automatically **calculated** based on the used bundle file names (they are **concatenated** and **hashed**). Example:

```
````html  
<abp-style-bundle>  
    <abp-style src="/libs/bootstrap/css/bootstrap.css" />  
    <abp-style src="/libs/font-awesome/css/font-awesome.css" />  
    <abp-style src="/libs/toastr/toastr.css" />  
    @if (ViewBag.IncludeCustomStyles != false)  
    {  
        <abp-style src="/styles/my-global-style.css" />  
    }  
</abp-style-bundle>  
``` `
```

This will potentially create **two different bundles** (one includes the `my-global-style.css` and other does not).

Advantages of **unnamed** bundles:

- \* Can **conditionally add items** to the bundle. But this may lead to multiple variations of the bundle based on the conditions.

Advantages of **named** bundles:

- \* Other **modules can contribute** to the bundle by its name (see the sections below).

### ### Single File

If you need to just add a single file to the page, you can use the `abp-script` or `abp-style` tag without a wrapping in the `abp-script-bundle` or `abp-style-bundle` tag.

Example:

```
````xml
```

```
<abp-script src="/scripts/my-script.js" />
````
```

The bundle name will be *\*scripts.my-scripts\** for the example above ("/" is replaced by "."). All bundling features are work as expected for single file bundles too.

## ## Bundling Options

If you need to use same bundle in **multiple pages** or want to use some more **powerful features**, you can configure bundles **by code** in your [module](./Module-Development-Basics.md) class.

### ### Creating A New Bundle

Example usage:

```
```C#
[DependsOn(typeof(AbpAspNetCoreMvcUiBundlingModule))]
public class MyWebModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        Configure<AbpBundlingOptions>(options =>
        {
            options
                .ScriptBundles
                .Add("MyGlobalBundle", bundle => {
                    bundle.AddFiles(
                        "/libs/jquery/jquery.js",
                        "/libs/bootstrap/js/bootstrap.js",
                        "/libs/toastr/toastr.min.js",
                        "/scripts/my-global-scripts.js"
                    );
                });
        });
    }
}
````
```

➤ You can use the same name (*\*MyGlobalBundle\** here) for a script & style bundle since they are added to different collections (``ScriptBundles`` and `StyleBundles`).

After defining such a bundle, it can be included into a page using the same tag helpers defined above. Example:

```
```html
<abp-script-bundle name="MyGlobalBundle" />
````
```

This time, no file defined in the tag helper definition because the bundle files are defined by the code.

#### ### Configuring An Existing Bundle

ABP supports [modularity](../../Module-Development-Basics.md) for bundling as well. A module can modify an existing bundle that is created by a depended module. Example:

```
~~~~C#
[DependsOn(typeof(MyWebModule))]
public class MyWebExtensionModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        Configure<AbpBundlingOptions>(options =>
        {
            options
                .ScriptBundles
                .Configure("MyGlobalBundle", bundle => {
                    bundle.AddFiles(
                        "/scripts/my-extension-script.js"
                    );
                });
        });
    }
}
~~~~
```

You can also use the `ConfigureAll` method to configure all existing bundles:

```
~~~~C#
[DependsOn(typeof(MyWebModule))]
public class MyWebExtensionModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        Configure<AbpBundlingOptions>(options =>
        {
            options
                .ScriptBundles
                .ConfigureAll(bundle => {
                    bundle.AddFiles(
                        "/scripts/my-extension-script.js"
                    );
                });
        });
    }
}
~~~~
```

#### ## Bundle Contributors

Adding files to an existing bundle seems useful. What if you need to **\*\*replace\*\*** a file in the bundle or you want to **\*\*conditionally\*\*** add files? Defining a bundle contributor provides extra power for such cases.

An example bundle contributor that replaces bootstrap.css with a customized version:

```
```C#
public class MyExtensionGlobalStyleContributor : BundleContributor
{
    public override void ConfigureBundle(BundleConfigurationContext context)
    {
        context.Files.ReplaceOne(
            "/libs/bootstrap/css/bootstrap.css",
            "/styles/extensions/bootstrap-customized.css"
        );
    }
}
````
```

Then you can use this contributor as like below:

```
```C#
services.Configure<AbpBundlingOptions>(options =>
{
    options
        .ScriptBundles
        .Configure("MyGlobalBundle", bundle => {
            bundle.AddContributors(typeof(MyExtensionGlobalStyleContributor));
        });
})
````
```

> You can also add contributors while creating a new bundle.

Contributors can also be used in the bundle tag helpers. Example:

```
```xml
<abp-style-bundle>
    <abp-style type="@typeof(BootstrapStyleContributor)" />
    <abp-style src="/libs/font-awesome/css/font-awesome.css" />
    <abp-style src="/libs/toastr/toastr.css" />
</abp-style-bundle>
````
```

``abp-style`` and ``abp-script`` tags can get ``type`` attributes (instead of ``src`` attributes) as shown in this sample. When you add a bundle contributor, its dependencies are also automatically added to the bundle.

### ### Contributor Dependencies

A bundle contributor can have one or more dependencies to other contributors.  
Example:

```
```C#
[DependsOn(typeof(MyDependedBundleContributor))] //Define the dependency
````
```

```

public class MyExtensionStyleBundleContributor : BundleContributor
{
 //...
}
```

```

When a bundle contributor is added, its dependencies are ****automatically and recursively**** added. Dependencies added by the ****dependency order**** by preventing ****duplicates****.

Duplicates are prevented even if they are in separated bundles. ABP organizes all bundles in a page and eliminates duplications.

Creating contributors and defining dependencies is a way of organizing bundle creation across different modules.

Contributor Extensions

In some advanced scenarios, you may want to do some additional configuration whenever a bundle contributor is used. Contributor extensions works seamlessly when the extended contributor is used.

The example below adds some styles for prism.js library:

```

```csharp
public class MyPrismjsStyleExtension : BundleContributor
{
 public override void ConfigureBundle(BundleConfigurationContext context)
 {
 context.Files.AddIfNotContains("/libs/prismjs/plugins/toolbar/prism-toolbar.css");
 }
}
```

```

Then you can configure ``AbpBundleContributorOptions`` to extend existing ``PrismjsStyleBundleContributor``.

```

```csharp
Configure<AbpBundleContributorOptions>(options =>
{
 options
 .Extensions<PrismjsStyleBundleContributor>()
 .Add<MyPrismjsStyleExtension>();
});
```

```

Whenever ``PrismjsStyleBundleContributor`` is added into a bundle, ``MyPrismjsStyleExtension`` will also be automatically added.

Accessing to the IServiceProvider

While it is rarely needed, ``BundleConfigurationContext`` has a ``ServiceProvider`` property that you can resolve service dependencies inside the ``ConfigureBundle`` method.

Standard Package Contributors

Adding a specific NPM package resource (js, css files) into a bundle is pretty straight forward for that package. For example you always add the `bootstrap.css` file for the bootstrap NPM package.

There are built-in contributors for all [standard NPM packages](Client-Side-Package-Management.md). For example, if your contributor depends on the bootstrap, you can just declare it, instead of adding the bootstrap.css yourself.

```
```c#
[DependsOn(typeof(BootstrapStyleContributor))] //Define the bootstrap style dependency
public class MyExtensionStyleBundleContributor : BundleContributor
{
 //...
}
```

Using the built-in contributors for standard packages;

- \* Prevents you typing **\*\*the invalid resource paths\*\***.
- \* Prevents changing your contributor if the resource **\*\*path changes\*\*** (the dependant contributor will handle it).
- \* Prevents multiple modules adding the **\*\*duplicate files\*\***.
- \* Manages **\*\*dependencies recursively\*\*** (adds dependencies of dependencies, if necessary).

### #### Volo.Abp.AspNetCore.Mvc.UI.Packages Package

➤ This package is already installed by default in the startup templates. So, most of the time, you don't need to install it manually.

If you're not using a startup template, you can use the [ABP CLI](../../CLI.md) to install it to your project. Execute the following command in the folder that contains the .csproj file of your project:

```
```
abp add-package Volo.Abp.AspNetCore.Mvc.UI.Packages
````
```

➤ If you haven't done it yet, you first need to install the [ABP CLI](../../CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.AspNetCore.Mvc.UI.Packages).

### ### Bundle Inheritance

In some specific cases, it may be needed to create a **\*\*new\*\*** bundle **\*\*inherited\*\*** from other bundle(s). Inheriting from a bundle (recursively) inherits all files/contributors of that bundle. Then the derived bundle can add or modify files/contributors **\*\*without modifying\*\*** the original bundle.

Example:

```
```c#
```

```

services.Configure<AbpBundlingOptions>(options =>
{
    options
        .StyleBundles
        .Add("MyTheme.MyGlobalBundle", bundle => {
            bundle
                .AddBaseBundles("MyGlobalBundle") //Can add multiple
                .AddFiles(
                    "/styles/mytheme-global-styles.css"
                );
        });
});
```

```

#### ## Additional Options

This section shows other useful options for the bundling and minification system.

#### #### Bundling Mode

ABP adds bundle files individually to the page for the `development` environment. It automatically bundles & minifies for other environments (`staging`, `production`...). Most of the times this is the behavior you would want. However, you may want to manually configure it in some cases. There are four modes;

- \* `Auto`: Automatically determines the mode based on the environment.
- \* `None`: No bundling or minification.
- \* `Bundle`: Bundled but not minified.
- \* `BundleAndMinify`: Bundled and minified.

You can configure `AbpBundlingOptions` in the `ConfigureServices` of your [module](`./Module-Development-Basics.md`).

#### **\*\*Example:\*\***

```

```csharp
Configure<AbpBundlingOptions>(options =>
{
    options.Mode = BundlingMode.Bundle;
});
```

```

#### ### Ignore For Minification

It is possible to ignore a specific file for the minification.

#### **\*\*Example:\*\***

```

```csharp
Configure<AbpBundlingOptions>(options =>
{
    options.MinificationIgnoredFiles.Add("/scripts/myscript.js");
}
```

```

```
});
```
```

Given file is still added to the bundle, but not minified in this case.

Load JavaScript and CSS asynchronously

You can configure `AbpBundlingOptions` to load all or a single js/css file asynchronously.

****Example:****

```
```csharp  
Configure<AbpBundlingOptions>(options =>
{
 options.PreloadStyles.Add("/__bundles/Basic.Global");
 options.DeferScriptsByDefault = true;
});
```  
  
**Output HTML:**  
```html  
<link rel="preload"
href="/__bundles/Basic.Global.F4FA61F368098407A4C972D0A6914137.css?v=637697363694828051"
as="style" onload="this.rel='stylesheet'"/>

<script defer
src="/libs/timeago/locales/jquery.timeago.en.js?v=637674729040000000"></script>
```
```

Themes

Themes uses the standard package contributors to add library resources to page layouts. Themes may also define some standard/global bundles, so any module can contribute to these standard/global bundles. See the [\[theming documentation\]](#) (Theming.md) for more.

Best Practices & Suggestions

It's suggested to define multiple bundles for an application, each one is used for different purposes.

- * ****Global bundle**:** Global style/script bundles are included to every page in the application. Themes already defines global style & script bundles. Your module can contribute to them.
- * ****Layout bundles**:** This is a specific bundle to an individual layout. Only contains resources shared among all the pages use the layout. Use the bundling tag helpers to create the bundle as a good practice.
- * ****Module bundles**:** For shared resources among the pages of an individual module.
- * ****Page bundles**:** Specific bundles created for each page. Use the bundling tag helpers to create the bundle as a best practice.

Establish a balance between performance, network bandwidth usage and count of many bundles.

See Also

- * [Client Side Package Management] (Client-Side-Package-Management.md)
- * [Theming] (Theming.md)

10.1.12 Tag Helpers

ABP Tag Helpers

ABP Framework defines a set of ****tag helper components**** to simply the user interface development for ASP.NET Core (MVC / Razor Pages) applications.

Bootstrap Component Wrappers

Most of the tag helpers are [Bootstrap] (<https://getbootstrap.com/>) (v5+) wrappers. Coding bootstrap is not so easy, not so type-safe and contains too much repetitive HTML tags. ABP Tag Helpers makes it ****easier**** and ****type safe****.

We don't aim to wrap bootstrap components 100%. Writing ****native bootstrap style code**** is still possible (actually, tag helpers generates native bootstrap code in the end), but we suggest to use the tag helpers wherever possible.

ABP Framework also adds some ****useful features**** to the standard bootstrap components.

Here, the list of components those are wrapped by the ABP Framework:

- * [Alerts] (Alerts.md)
- * [Badges] (Badges.md)
- * [Blockquote] (Blockquote.md)
- * [Borders] (Borders.md)
- * [Breadcrumb] (Breadcrumbs.md)
- * [Buttons] (Buttons.md)
- * [Cards] (Cards.md)
- * [Carousel] (Carousel.md)
- * [Collapse] (Collapse.md)
- * [Dropdowns] (Dropdowns.md)
- * [Figures] (Figure.md)
- * [Grids] (Grids.md)
- * [List Groups] (List-Groups.md)
- * [Modals] (Modals.md)
- * [Navigation] (Navs.md)
- * [Paginator] (Paginator.md)
- * [Popovers] (Popovers.md)
- * [Progress Bars] (Progress-Bars.md)
- * [Tables] (Tables.md)
- * [Tabs] (Tabs.md)
- * [Tooltips] (Tooltips.md)

➤ Until all the tag helpers are documented, you can visit <https://bootstrap-taghelpers.abp.io/> to see them with live samples.

Form Elements

****Abp Tag Helpers**** add new features to standard ****Asp. Net Core MVC input & select Tag Helpers**** and wrap them with ****Bootstrap**** form controls. See [[Form Elements documentation](#)](Form-elements.md) .

Dynamic Forms

****Abp Tag helpers**** offer an easy way to build complete ****Bootstrap forms****. See [[Dynamic Forms documentation](#)](Dynamic-Forms.md) .

10.1.12.1 Form Elements

Form Elements

Introduction

Abp provides form input tag helpers to make building forms easier.

Demo

See the [[form elements demo page](#)](<https://bootstrap-taghelpers.abp.io/Components/FormElements>) to see it in action.

abp-input

`abp-input` tag creates a Bootstrap form input for a given c# property. It uses [[Asp. Net Core Input Tag Helper](#)](<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-7.0#the-input-tag-helper>) in background, so every data annotation attribute of `input` tag helper of Asp. Net Core is also valid for `abp-input`.

Usage:

```
```xml
<abp-input asp-for="@Model.MyModel.Name"/>
<abp-input asp-for="@Model.MyModel.Description"/>
<abp-input asp-for="@Model.MyModel.Password"/>
<abp-input asp-for="@Model.MyModel.IsActive"/>
```
```

Model:

```
```csharp
public class FormElementsModel : PageModel
{
 public SampleModel MyModel { get; set; }
```

```

public void OnGet()
{
 MyModel = new SampleModel();
}

public class SampleModel
{
 [Required]
 [Placeholder("Enter your name...")]
 [InputInfoText("What is your name?")]
 public string Name { get; set; }

 [Required]
 [FormControlSize(AbpFormControlSize.Large)]
 public string SurName { get; set; }

 [TextArea(Rows = 4)]
 public string Description { get; set; }

 [Required]
 [DataType(DataType.Password)]
 public string Password { get; set; }

 public bool IsActive { get; set; }
}
```
.....

```

Attributes

You can set some of the attributes on your c# property, or directly on HTML tag. If you are going to use this property in a [\[abp-dynamic-form\]](#) (`Dynamic-forms.md`), then you can only set these properties via property attributes.

Property Attributes

- `[TextArea()]`: Converts the input into a text area.
- * `[Placeholder()]`: Sets the description are of the input. You can use a localization key directly.
- * `[InputInfoText()]`: Sets text for the input. You can directly use a localization key.
- * `[FormControlSize()]`: Sets the size of the form-control wrapper element. Available values are
 - `AbpFormControlSize.Default`
 - `AbpFormControlSize.Small`
 - `AbpFormControlSize.Medium`
 - `AbpFormControlSize.Large`
- * `[DisabledInput]` : Sets the input as disabled.
- * `[ReadOnlyInput]` : Sets the input as read-only.

Tag Attributes

- * `info`: Sets text for the input. You can directly use a localization key.
- * `auto-focus`: It lets the browser set focus to the element when its value is true.
- * `size`: Sets the size of the form-control wrapper element. Available values are
 - `AbpFormControlSize.Default`
 - `AbpFormControlSize.Small`
 - `AbpFormControlSize.Medium`
 - `AbpFormControlSize.Large`
- * `disabled`: Sets the input as disabled.
- * `readonly`: Sets the input as read-only.
- * `label`: Sets the label of input.
- * `required-symbol`: Adds the required symbol `(*)` to the label when the input is required. The default value is `True`.
- * `floating-label`: Sets the label as floating label. The default value is `False`.

`asp-format`, `name` and `value` attributes of [\[Asp.NET Core Input Tag Helper\]](#) (<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-7.0#the-input-tag-helper>) are also valid for `abp-input` tag helper.

Label & Localization

You can set the label of the input in several ways:

- You can use the `Label` attribute to set the label directly. This property does not automatically localize the text. To localize the label, use `label="@L["{LocalizationKey}"].Value"`.
- You can set it using ` [Display(name="{LocalizationKey}")]` attribute of ASP.NET Core.
- You can just let **abp** find the localization key for the property. It will try to find "DisplayName:{PropertyName}" or "{PropertyName}" localization keys, if `label` or ` [DisplayName]` attributes are not set.

abp-select

`abp-select` tag creates a Bootstrap form select for a given c# property. It uses [\[ASP.NET Core Select Tag Helper\]](#) (<https://docs.microsoft.com/tr-tr/aspnet/core/mvc/views/working-with-forms?view=aspnetcore-3.1#the-select-tag-helper>) in background, so every data annotation attribute of `select` tag helper of ASP.NET Core is also valid for `abp-select`.

`abp-select` tag needs a list of `Microsoft.AspNetCore.Mvc.Rendering.SelectListItem` to work. It can be provided by `asp-items` attriube on the tag or ` [SelectItems()]` attribute on c# property. (if you are using [\[abp-dynamic-form\]](#) (Dynamic-forms.md), c# attribute is the only way.)

`abp-select` supports multiple selection.

`abp-select` auto-creates a select list for **Enum** properties. No extra data is needed. If property is nullable, an empty key and value is added to top of the auto-generated list.

Usage:

```
```xml
```

```

<abp-select asp-for="@Model.MyModel.City" asp-items="@Model.CityList"/>

<abp-select asp-for="@Model.MyModel.AnotherCity"/>

<abp-select asp-for="@Model.MyModel.MultipleCities" asp-items="@Model.CityList"/>

<abp-select asp-for="@Model.MyModel.MyCarType"/>

<abp-select asp-for="@Model.MyModel.MyNullableCarType"/>
````
```

Model:

```

```csharp
public class FormElementsModel : PageModel
{
 public SampleModel MyModel { get; set; }

 public List<SelectListItem> CityList { get; set; }

 public void OnGet()
 {
 MyModel = new SampleModel();

 CityList = new List<SelectListItem>
 {
 new SelectListItem { Value = "NY", Text = "New York" },
 new SelectListItem { Value = "LDN", Text = "London" },
 new SelectListItem { Value = "IST", Text = "Istanbul" },
 new SelectListItem { Value = "MOS", Text = "Moscow" }
 };
 }
}

public class SampleModel
{
 public string City { get; set; }

 [SelectItems(nameof(CityList))]
 public string AnotherCity { get; set; }

 public List<string> MultipleCities { get; set; }

 public CarType MyCarType { get; set; }

 public CarType? MyNullableCarType { get; set; }
}

public enum CarType
{
 Sedan,
 Hatchback,
 StationWagon,
```

```
 Coupe
 }
...
....
```

### ### Attributes

You can set some of the attributes on your c# property, or directly on HTML tag. If you are going to use this property in a [abp-dynamic-form] (Dynamic-forms.md), then you can only set these properties via property attributes.

#### #### Property Attributes

- \* `#[SelectItems()]` : Sets the select data. Parameter should be the name of the data list. (see example above)
- `#[InputInfoText()]` : Sets text for the input. You can directly use a localization key.
- `#[FormControlSize()]` : Sets the size of the form-control wrapper element. Available values are
  - `AwpFormControlSize.Default`
  - `AwpFormControlSize.Small`
  - `AwpFormControlSize.Medium`
  - `AwpFormControlSize.Large`

#### #### Tag Attributes

- `asp-items` : Sets the select data. This Should be a list of SelectListItem.
- `info` : Sets text for the input. You can directly use a localization key.
- `size` : Sets the size of the form-control wrapper element. Available values are
  - `AwpFormControlSize.Default`
  - `AwpFormControlSize.Small`
  - `AwpFormControlSize.Medium`
  - `AwpFormControlSize.Large`
- `label` : Sets the label of input.
- `required-symbol` : Adds the required symbol `(\*)` to the label when the input is required. The default value is `True` .

#### ### Label & Localization

You can set the label of the input in several ways:

- You can use `Label` attribute and directly set the label. But it doesn't auto localize your localization key. So use it as `label="@L["{LocalizationKey}"].Value".`
- You can set it using `#[Display(name="{LocalizationKey}")]` attribute of ASP.NET Core.
- You can just let \*\*abp\*\* find the localization key for the property. It will try to find "DisplayName:{PropertyName}" or "{PropertyName}" localization keys.

Localizations of combobox values are set by `abp-select` for \*\*Enum\*\* property. It searches for "{EnumType Name}. {Enum Property Name}" or "{Enum Property Name}" localization keys. For instance, in the example above, it will use "CarType.StationWagon" or "StationWagon" keys for localization when it localizes combobox values.

```
abp-radio
```

``abp-radio`` tag creates a Bootstrap form radio group for a given c# property. Usage is very similar to ``abp-select`` tag.

Usage:

```
```xml
<abp-radio asp-for="@Model.MyModel.CityRadio" asp-items="@Model.CityList" inline="true"/>

<abp-radio asp-for="@Model.MyModel.CityRadio2"/>
````
```

Model:

```
```csharp
public class FormElementsModel : PageModel
{
    public SampleModel MyModel { get; set; }

    public List<SelectListItem> CityList { get; set; } = new List<SelectListItem>
    {
        new SelectListItem { Value = "NY", Text = "New York" },
        new SelectListItem { Value = "LDN", Text = "London" },
        new SelectListItem { Value = "IST", Text = "Istanbul" },
        new SelectListItem { Value = "MOS", Text = "Moscow" }
    };

    public void OnGet()
    {
        MyModel = new SampleModel();
        MyModel.CityRadio = "IST";
        MyModel.CityRadio2 = "MOS";
    }
}

public class SampleModel
{
    public string CityRadio { get; set; }

    [SelectItems(nameof(CityList))]
    public string CityRadio2 { get; set; }
}
````
```

### ### Attributes

You can set some of the attributes on your c# property, or directly on HTML tag. If you are going to use this property in a [`abp-dynamic-form`](Dynamic-forms.md), then you can only set these properties via property attributes.

#### #### Property Attributes

- `@[SelectItems()]`: Sets the select data. Parameter should be the name of the data list. (see example above)

#### #### Tag Attributes

- `asp-items`: Sets the select data. This Should be a list of SelectListItem.
- `Inline`: If true, radio buttons will be in single line, next to each other. If false, they will be under each other.

#### ## abp-date-picker & abp-date-range-picker

`abp-date-picker` and `abp-date-range-picker` tags creates a Bootstrap form date picker for a given c# property. `abp-date-picker` is for single date selection, `abp-date-range-picker` is for date range selection. It uses [datepicker](<https://www.daterangepicker.com/>) jQuery plugin.

Usage:

```
```xml
<abp-date-picker asp-for="@Model.MyModel.MyDate" />
<abp-date-range-picker asp-for-start="@Model.MyModel.MyDateRangeStart" asp-for-end="@Model.MyModel.MyDateRangeEnd" />
<abp-dynamic-form abp-model="DynamicFormExample"></abp-dynamic-form>
```
```

Model:

```
```csharp
public class FormElementsModel : PageModel
{
    public SampleModel MyModel { get; set; }

    public DynamicForm DynamicFormExample { get; set; }

    public void OnGet()
    {
        MyModel = new SampleModel();

        DynamicFormExample = new DynamicForm();
    }

    public class SampleModel
    {
        public DateTime MyDate { get; set; }

        public DateTime MyDateRangeStart { get; set; }

        public DateTime MyDateRangeEnd { get; set; }
    }

    public class DynamicForm
```

```

{
    [DateRangePicker("MyPicker", true)]
    public DateTime StartDate { get; set; }

    [DateRangePicker("MyPicker", false)]
    [DatePickerOptions(nameof(DatePickerOptions))]
    public DateTime EndDate { get; set; }

    public DateTime DateTime { get; set; }

    public DynamicForm()
    {
        StartDate = DateTime.Now;
        EndDate = DateTime.Now;
        DateTime = DateTime.Now;
    }
}

public AbpDatePickerOptions DatePickerOptions { get; set; }
```

```

### ### Attributes

You can set some of the attributes on your c# property, or directly on HTML tag. If you are going to use this property in a `[abp-dynamic-form]` (`Dynamic-forms.mdd`), then you can only set these properties via property attributes.

#### #### Property Attributes

- \* `^[Placeholder()]``: Sets the description are of the input. You can use a localization key directly.
- \* `^[InputInfoText()]``: Sets text for the input. You can directly use a localization key.
- \* `^[FormControlSize()]``: Sets the size of the form-control wrapper element. Available values are
  - `AbpFormControlSize.Default``
  - `AbpFormControlSize.Small``
  - `AbpFormControlSize.Medium``
  - `AbpFormControlSize.Large``
- \* `^[DisabledInput]`` : Sets the input as disabled.
- \* `^[ReadOnlyInput]`` : Sets the input as read-only.
- `^[DatePickerOptions()]``: Sets the predefined options of the date picker. Parameter should be the name of the options property (see example above). See the available `[datepicker options]` (<https://www.daterangepicker.com/#options>). You can use a localization key directly.

#### ##### abp-date-picker

`^[DatePicker]`` : Sets the input as datepicker. Especially for string properties.

#### ##### abp-date-range-picker

`[DateRangePicker()]` : Sets the picker id for the date range picker. You can set the property as a start date by setting IsStart=true or leave it as default/false to set it as an end date.

#### #### Tag Attributes

- \* `info` : Sets text for the input. You can directly use a localization key.
- \* `auto-focus` : It lets the browser set focus to the element when its value is true.
- \* `size` : Sets the size of the form-control wrapper element. Available values are
  - `AwpFormControlSize.Default`
  - `AwpFormControlSize.Small`
  - `AwpFormControlSize.Medium`
  - `AwpFormControlSize.Large`
- \* `disabled` : Sets the input as disabled.
- \* `readonly` : Sets the input as read-only.
- \* `label` : Sets the label of input.
- \* `required-symbol` : Adds the required symbol `(\*)` to the label when the input is required. The default value is `True`.
- \* `open-button` : A button to open the datepicker will be added when its `True`. The default value is `True`.
- \* `clear-button` : A button to clear the datepicker will be added when its `True`. The default value is `True`.
- \* `single-open-and-clear-button` : Shows the open and clear buttons in a single button when it's `True`. The default value is `True`.
- \* `is-utc` : Converts the date to UTC when its `True`. The default value is `False`.
- \* `is-iso` : Converts the date to ISO format when its `True`. The default value is `False`.
- \* `date-format` : Sets the date format of the input. The default format is the user's culture date format. You need to provide a JavaScript date format convention. Eg: `YYYY-MM-DDTHH:MM:SSZ`.
- \* `date-separator` : Sets a character to separate start and end dates. The default value is `\_`.
- \* Other non-mapped attributes will be automatically added to the input element as is. See the available [datepicker options](<https://www.daterangepicker.com/#options>). Eg: `data-start-date="2020-01-01"

#### ##### abp-date-picker

- \* `asp-date` : Sets the date value. This should be a `DateTime`, `DateTime?`, `DateTimeOffset`, `DateTimeOffset?` or `string` value.

#### ##### abp-date-range-picker

- \* `asp-for-start` : Sets the start date value. This should be a `DateTime`, `DateTime?`, `DateTimeOffset`, `DateTimeOffset?` or `string` value.
- \* `asp-for-end` : Sets the end date value. This should be a `DateTime`, `DateTime?`, `DateTimeOffset`, `DateTimeOffset?` or `string` value.

### ## Label & Localization

You can set the label of the input in several ways:

- You can use the `Label` attribute to set the label directly. This property does not automatically localize the text. To localize the label, use `label="@L[{LocalizationKey}].Value"`.
- You can set it using `[Display(name="{LocalizationKey}")]` attribute ASP.NET Core.
- You can just let \*\*abp\*\* find the localization key for the property. If the `label` or `[DisplayName]` attributes are not set, it tries to find the localization by convention. For example `DisplayName:{YourPropertyName}` or `{YourPropertyName}`. If your property name is FullName, it will search for `DisplayName:FullName` or `{FullName}`.

#### #### JavaScript Usage

```
```javascript
var newPicker = abp.libs.bootstrapDatePicker.createDateRangePicker(
{
    label: "New Picker",
}
);
newPicker.insertAfter($('body'));
```

```javascript
var newPicker = abp.libs.bootstrapDatePicker.createSinglePicker(
{
    label: "New Picker",
}
);
newPicker.insertAfter($('body'));
```

```

#### #### Options

- \* `label`: Sets the label of the input.
- \* `placeholder`: Sets the placeholder of the input.
- \* `value`: Sets the value of the input.
- \* `name`: Sets the name of the input.
- \* `id`: Sets the id of the input.
- \* `required`: Sets the input as required.
- \* `disabled`: Sets the input as disabled.
- \* `readonly`: Sets the input as read-only.
- \* `size`: Sets the size of the form-control wrapper element. Available values are
  - `AbpFormControlSize.Default`
  - `AbpFormControlSize.Small`
  - `AbpFormControlSize.Medium`
  - `AbpFormControlSize.Large`
- \* `openButton`: A button to open the datepicker will be added when its `True`. The default value is `True`.
- \* `clearButton`: A button to clear the datepicker will be added when its `True`. The default value is `True`.
- \* `singleOpenAndClearButton`: Shows the open and clear buttons in a single button when it's `True`. The default value is `True`.
- \* `isUtc`: Converts the date to UTC when its `True`. The default value is `False`.
- \* `isIso`: Converts the date to ISO format when its `True`. The default value is `False`.

- \* `dateFormat`: Sets the date format of the input. The default format is the user's culture date format. You need to provide a JavaScript date format convention. Eg: `YYYY-MM-DDTHH:MM:SSZ`.
- \* `dateSeparator`: Sets a character to separate start and end dates. The default value is `.`
- \* `startDateName`: Sets the name of the hidden start date input.
- \* `endDateName`: Sets the name of the hidden end date input.
- \* `dateName`: Sets the name of the hidden date input.
- \* Other [datepicker options](https://www.daterangepicker.com/#options). Eg: `startDate: "2020-01-01"`.

### 10.1.12.2 Dynamic Forms

```
Dynamic Forms
```

```
Introduction
```

`abp-dynamic-form` creates a bootstrap form for a given c# model.

Basic usage:

```
```xml
<abp-dynamic-form abp-model="@Model.MyDetailedModel"/>
````
```

Model:

```
```csharp
public class DynamicFormsModel : PageModel
{
    [BindProperty]
    public DetailedModel MyDetailedModel { get; set; }

    public List<SelectListItem> CountryList { get; set; } = new List<SelectListItem>
    {
        new SelectListItem { Value = "CA", Text = "Canada" },
        new SelectListItem { Value = "US", Text = "USA" },
        new SelectListItem { Value = "UK", Text = "United Kingdom" },
        new SelectListItem { Value = "RU", Text = "Russia" }
    };

    public void OnGet()
    {
        MyDetailedModel = new DetailedModel
        {
            Name = "",
            Description = "Lorem ipsum dolor sit amet.",
            IsActive = true,
            Age = 65,
            Day = DateTime.Now,
            MyCarType = CarType.Coupe,
```

```

        YourCarType = CarType.Sedan,
        Country = "RU",
        NeighborCountries = new List<string>() { "UK", "CA" }
    } ;
}

public class DetailedModel
{
    [Required]
    [Placeholder("Enter your name...")]
    [Display(Name = "Name")]
    public string Name { get; set; }

    [TextArea(Rows = 4)]
    [Display(Name = "Description")]
    [InputInfoText("Describe Yourself")]
    public string Description { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Password")]
    public string Password { get; set; }

    [Display(Name = "Is Active")]
    public bool IsActive { get; set; }

    [Required]
    [Display(Name = "Age")]
    public int Age { get; set; }

    [Required]
    [Display(Name = "My Car Type")]
    public CarType MyCarType { get; set; }

    [Required]
    [AbpRadioButton(Inline = true)]
    [Display(Name = "Your Car Type")]
    public CarType YourCarType { get; set; }

    [DataType(DataType.Date)]
    [Display(Name = "Day")]
    public DateTime Day { get; set; }

    [SelectItems(nameof(CountryList))]
    [Display(Name = "Country")]
    public string Country { get; set; }

    [SelectItems(nameof(CountryList))]
    [Display(Name = "Neighbor Countries")]
    public List<string> NeighborCountries { get; set; }
}

```

```
public enum CarType
{
    Sedan,
    Hatchback,
    StationWagon,
    Coupe
}
```

Demo

See the [dynamic forms demo page](<https://bootstrap-taghelpers.abp.io/Components/DynamicForms>) to see it in action.

Attributes

abp-model

Sets the c# model for dynamic form. Properties of this modal are turned into inputs in the form.

column-size

Here, use 'col-sm' to set the size . When setting this property 'col-12' will be added at the same time.

submit-button

Can be `True` or `False`.

If `True` , a submit button will be generated at the bottom of the form.

Default value is `False` .

required-symbols

Can be `True` or `False` .

If `True` , required inputs will have a symbol (*) that indicates they are required.

Default value is `True` .

Form Content Placement

By default, `abp-dynamic-form` clears the inner html and places the inputs into itself. If you want to add additional content to dynamic form or place the inputs to some specific area, you can use `<abp-form-content />` tag. This tag will be replaced by form content and rest of the inner html of `abp-dynamic-form` tag will be unchanged.

Usage:

```

```xml
<abp-dynamic-form abp-model="@Model.MyExampleModel">
 <div>
 Some content....
 </div>
 <div class="input-area">
 <abp-form-content />
 </div>
 <div>
 Some more content....
 </div>
</abp-dynamic-form>
```

```

Input Order

`abp-dynamic-form` orders the properties by their `DisplayOrder` attribute and then their property order in model class.

Default `DisplayOrder` attribute number is 10000 for every property.

See example below:

```

```csharp
public class OrderExampleModel
{
 [DisplayOrder(10004)]
 public string Name{ get; set; }

 [DisplayOrder(10005)]
 public string Surname{ get; set; }

 //Default 10000
 public string EmailAddress { get; set; }

 [DisplayOrder(10003)]
 public string PhoneNumber { get; set; }

 [DisplayOrder(9999)]
 public string City { get; set; }
}
```

```

In this example, input fields will be displayed with this order: `City` > `EmailAddress` > `PhoneNumber` > `Name` > `Surname`.

Ignoring a property

By default, `abp-dynamic-form` generates input for every property in model class. If you want to ignore a property, use `DynamicFormIgnore` attribute.

See example below:

```

```csharp
public class MyModel
{
 public string Name { get; set; }

 [DynamicFormIgnore]
 public string Surname { get; set; }
}
```

```

In this example, no input will be generated for `Surname` property.

Indicating Text box, Radio Group and Combobox

If you have read the [Form elements document](Form-elements.md), you noticed that `abp-radio` and `abp-select` tags are very similar on c# model. So we have to use `[AbpRadioButton()]` attribute to tell `abp-dynamic-form` which of your properties will be radio group and which will be combobox. See example below:

```

```xml
<abp-dynamic-form abp-model="@Model.MyDetailedModel"/>
```

```

Model:

```

```csharp
public class DynamicFormsModel : PageModel
{
 [BindProperty]
 public DetailedModel MyDetailedModel { get; set; }

 public List<SelectListItem> CountryList { get; set; } = new List<SelectListItem>
 {
 new SelectListItem { Value = "CA", Text = "Canada" },
 new SelectListItem { Value = "US", Text = "USA" },
 new SelectListItem { Value = "UK", Text = "United Kingdom" },
 new SelectListItem { Value = "RU", Text = "Russia" }
 };

 public void OnGet()
 {
 MyDetailedModel = new DetailedModel
 {
 ComboCarType = CarType.Coupe,
 RadioCarType = CarType.Sedan,
 ComboCountry = "RU",
 RadioCountry = "UK"
 };
 }

 public class DetailedModel
 {
 }
}
```

```

```

{
    public CarType ComboCarType { get; set; }

    [AbpRadioButton(Inline = true)]
    public CarType RadioCarType { get; set; }

    [SelectItems(nameof(CountryList))]
    public string ComboCountry { get; set; }

    [AbpRadioButton()]
    [SelectItems(nameof(CountryList))]
    public string RadioCountry { get; set; }
}

public enum CarType
{
    Sedan,
    Hatchback,
    StationWagon,
    Coupe
}
```

```

As you see in example above:

- \* If ` `[AbpRadioButton()]` are used on a **\*\*Enum\*\*** property, it will be a radio group. Otherwise, combobox.
- \* If ` `[SelectItems()]` and ` `[AbpRadioButton()]` are used on a property, it will be a radio group.
- \* If just ` `[SelectItems()]` is used on a property, it will be a combobox.
- \* If none of these attributes are used on a property, it will be a text box.

#### ## Localization

` `abp-dynamic-form` handles localization as well.

By default, it will try to find "DisplayName:{PropertyName}" or "{PropertyName}" localization keys and set the localization value as input label.

You can set it yourself by using ` `[Display()]` attribute of Asp.Net Core. You can use a localization key in this attribute. See example below:

```

```csharp
[Display(Name = "Name")]
public string Name { get; set; }
```

```

#### ## See Also

- \* [\[Form Elements\]](#) (Form-elements.md)

## 10.1.13 Widgets

```
Widgets
```

ABP provides a model and infrastructure to create **\*\*reusable widgets\*\***. Widget system is an extension to [ASP.NET Core's ViewComponents] (<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/view-components>). Widgets are especially useful when you want to;

- \* Have **\*\*scripts & styles\*\*** dependencies for your widget.
- \* Create **\*\*dashboards\*\*** with widgets used inside.
- \* Define widgets in reusable **\*\*[modules](../../Module-Development-Basics.md)\*\***.
- \* Co-operate widgets with **\*\*[authorization](../../Authorization.md)\*\*** and **\*\*[bundling](Bundling-Minification.md)\*\*** systems.

```
Basic Widget Definition
```

```
Create a View Component
```

As the first step, create a new regular ASP.NET Core View Component:

```
![widget-basic-files](../../images/widget-basic-files.png)
```

**\*\*MySimpleWidgetViewComponent.cs\*\*:**

```
~~~~csharp
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;

namespace DashboardDemo.Web.Pages.Components.MySimpleWidget
{
    public class MySimpleWidgetViewComponent : AbpViewComponent
    {
        public IViewComponentResult Invoke()
        {
            return View();
        }
    }
}~~~
```

Inheriting from `AbpViewComponent` is not required. You could inherit from ASP.NET Core's standard `ViewComponent`. `AbpViewComponent` only defines some base useful properties.

You can inject a service and use in the `Invoke` method to get some data from the service. You may need to make Invoke method async, like `public async Task<IViewComponentResult> InvokeAsync()`. See [ASP.NET Core's ViewComponents] (<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/view-components>) document fore all different usages.

**\*\*Default.cshtml\*\*:**

```
~~~xml
```

```
<div class="my-simple-widget">
 <h2>My Simple Widget</h2>
 <p>This is a simple widget!</p>
</div>
````
```

Define the Widget

Add a `Widget` attribute to the `MySimpleWidgetViewComponent` class to mark this view component as a widget:

```
```csharp
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Widgets;

namespace DashboardDemo.Web.Pages.Components
{
 [Widget]
 public class MySimpleWidgetViewComponent : AbpViewComponent
 {
 public IViewComponentResult Invoke()
 {
 return View();
 }
 }
}
````
```

Rendering a Widget

Rendering a widget is pretty standard. Use the `Component.InvokeAsync` method in a razor view/page as you do for any view component. Examples:

```
```xml
@await Component.InvokeAsync("MySimpleWidget")
@await Component.InvokeAsync(typeof(MySimpleWidgetViewComponent))
````
```

First approach uses the widget name while second approach uses the view component type.

Widgets with Arguments

ASP.NET Core's view component system allows you to accept arguments for view components. The sample view component below accepts `startDate` and `endDate` and uses these arguments to retrieve data from a service.

```
```csharp
using System;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
````
```

```

using Volo.Abp.AspNetCore.Mvc.UI.Widgets;

namespace DashboardDemo.Web.Pages.Shared.Components.CountersWidget
{
    [Widget]
    public class CountersWidgetViewComponent : AbpViewComponent
    {
        private readonly IDashboardAppService _dashboardAppService;

        public CountersWidgetViewComponent(IDashboardAppService dashboardAppService)
        {
            _dashboardAppService = dashboardAppService;
        }

        public async Task<IViewComponentResult> InvokeAsync(
            DateTime startDate, DateTime endDate)
        {
            var result = await _dashboardAppService.GetCountersWidgetAsync(
                new CountersWidgetInputDto
                {
                    StartDate = startDate,
                    EndDate = endDate
                }
            );

            return View(result);
        }
    }
}
```

```

Now, you need to pass an anonymous object to pass arguments as shown below:

```

```xml
@await Component.InvokeAsync("CountersWidget", new
{
    startDate = DateTime.Now.Subtract(TimeSpan.FromDays(7)),
    endDate = DateTime.Now
})
```

```

`## Widget Name`

Default name of the view components are calculated based on the name of the view component type. If your view component type is `MySimpleWidgetViewComponent` then the widget name will be `MySimpleWidget` (removes `ViewComponent` postfix). This is how ASP.NET Core calculates a view component's name.

To customize widget's name, just use the standard `ViewComponent` attribute of ASP.NET Core:

```

```csharp

```

```

using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Widgets;

namespace DashboardDemo.Web.Pages.Components.MySimpleWidget
{
    [Widget]
    [ViewComponent(Name = "MyCustomNamedWidget")]
    public class MySimpleWidgetViewComponent : AbpViewComponent
    {
        public IViewComponentResult Invoke()
        {
            return View("~/Pages/Components/MySimpleWidget/Default.cshtml");
        }
    }
}
```

```

ABP will respect to the custom name by handling the widget.

- If the view component name and the folder name of the view component don't match, you may need to manually write the view path as done in this example.

#### #### Display Name

You can also define a human-readable, localizable display name for the widget. This display name then can be used on the UI when needed. Display name is optional and can be defined using properties of the `Widget` attribute:

```

```csharp
using DashboardDemo.Localization;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Widgets;

namespace DashboardDemo.Web.Pages.Components.MySimpleWidget
{
    [Widget(
        DisplayName = "MySimpleWidgetDisplayName", //Localization key
        DisplayNameResource = typeof(DashboardDemoResource) //localization resource
    )]
    public class MySimpleWidgetViewComponent : AbpViewComponent
    {
        public IViewComponentResult Invoke()
        {
            return View();
        }
    }
}
```

```

See [[the localization document](#)](..../Localization.md) to learn about localization resources and keys.

## ## Style & Script Dependencies

There are some challenges when your widget has script and style files;

- \* Any page uses the widget should also include the **\*\*its script & styles\*\*** files into the page.
- \* The page should also care about **\*\*depended libraries/files\*\*** of the widget.

ABP solves these issues when you properly relate the resources with the widget. You don't care about dependencies of the widget while using it.

## ### Defining as Simple File Paths

The example widget below adds a style and a script file:

```
```csharp
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Widgets;

namespace DashboardDemo.Web.Pages.Components.MySimpleWidget
{
    [Widget(
        StyleFiles = new[] { "/Pages/Components/MySimpleWidget/Default.css" },
        ScriptFiles = new[] { "/Pages/Components/MySimpleWidget/Default.js" }
    )]
    public class MySimpleWidgetViewComponent : AbpViewComponent
    {
        public IViewComponentResult Invoke()
        {
            return View();
        }
    }
}
````
```

ABP takes account these dependencies and properly adds to the view/page when you use the widget. Style/script files can be **\*\*physical or virtual\*\***. It is completely integrated to the [[Virtual File System](#)](..../Virtual-File-System.md).

## ### Defining Bundle Contributors

All resources for used widgets in a page are added as a **\*\*bundle\*\*** (bundled & minified in production if you don't configure otherwise). In addition to adding a simple file, you can take full power of the bundle contributors.

The sample code below does the same with the code above, but defines and uses bundle contributors:

```

```csharp
using System.Collections.Generic;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc;
using Volo.Abp.AspNetCore.Mvc.UI.Bundling;
using Volo.Abp.AspNetCore.Mvc.UI.Widgets;

namespace DashboardDemo.Web.Pages.Components.MySimpleWidget
{
    [Widget(
        StyleTypes = new []{ typeof(MySimpleWidgetStyleBundleContributor) },
        ScriptTypes = new[]{ typeof(MySimpleWidgetScriptBundleContributor) }
    )]
    public class MySimpleWidgetViewComponent : AbpViewComponent
    {
        public IViewComponentResult Invoke()
        {
            return View();
        }
    }

    public class MySimpleWidgetStyleBundleContributor : BundleContributor
    {
        public override void ConfigureBundle(BundleConfigurationContext context)
        {
            context.Files
                .AddIfNotContains("/Pages/Components/MySimpleWidget/Default.css");
        }
    }

    public class MySimpleWidgetScriptBundleContributor : BundleContributor
    {
        public override void ConfigureBundle(BundleConfigurationContext context)
        {
            context.Files
                .AddIfNotContains("/Pages/Components/MySimpleWidget/Default.js");
        }
    }
}
```

```

Bundle contribution system is very powerful. If your widget uses a JavaScript library to render a chart, then you can declare it as a dependency, so the JavaScript library is automatically added to the page if it wasn't added before. In this way, the page using your widget doesn't care about the dependencies.

See the [\[bundling & minification\]](#) ([Bundling-Minification.md](#)) documentation for more information about that system.

[## RefreshUrl](#)

A widget may design a `RefreshUrl` that is used whenever the widget needs to be refreshed. If it is defined, the widget is re-rendered on the server side on every refresh (see the refresh `method` of the `WidgetManager` below).

```
```csharp
[Widget(RefreshUrl = "Widgets/Counters")]
public class CountersWidgetViewComponent : AbpViewComponent
{
}

````
```

Once you define a `RefreshUrl` for your widget, you need to provide an endpoint to render and return it:

```
```csharp
[Route("Widgets")]
public class CountersWidgetController : AbpController
{
    [HttpGet]
    [Route("Counters")]
    public IActionResult Counters(DateTime startDate, DateTime endDate)
    {
        return ViewComponent("CountersWidget", new {startDate, endDate});
    }
}
````
```

`Widgets/Counters` route matches to the `RefreshUrl` declared before.

> A widget supposed to be refreshed in two ways: In the first way, when you use a `RefreshUrl`, it re-rendered on the server and replaced by the HTML returned from server. In the second way the widget gets data (generally a JSON object) from server and refreshes itself in the client side (see the refresh method in the Widget JavaScript API section).

## AutoInitialize

`WidgetAttribute` has an `AutoInitialize` property (`bool`) that can be set to automatically initialize a widget on page ready & whenever the widget is added to the DOM. The default value is `false`.

If a widget is configured to be auto initialized, then a `WidgetManager` (see below) automatically created and initialized for instances of this widget. This is useful when the widget instances are not grouped and separately works (they don't require to init or refresh together).

Setting the `AutoInitialize` to `true` is equivalent to write such a code yourself:

```
```js
$('.abp-widget-wrapper[data-widget-name="MySimpleWidget"]')
.each(function () {
    var widgetManager = new abp.WidgetManager({
````
```

```

 wrapper: $(this),
 });

 widgetManager.init($(this));
}
```

```

> `AutoInitialize` also supports widgets loaded/refreshed via AJAX (added to the DOM later) and/or used in a nested way (a widget inside another widget). If you don't need to group multiple widgets and control with a single `WidgetManager`, `AutoInitialize` is the recommended approach.

JavaScript API

A widget may need to be rendered and refreshed in the client side. In such cases, you can use ABP's `WidgetManager` and define APIs for your widgets.

WidgetManager

`WidgetManager` is used to initialize and refresh one or more widgets. Create a new `WidgetManager` as shown below:

```

```js
$(function() {
 var myWidgetManager = new abp.WidgetManager('#MyDashboardWidgetsArea');
})
```

```

`MyDashboardWidgetsArea` may contain one or more widgets inside.

> Using the `WidgetManager` inside document.ready (like above) is a good practice since its functions use the DOM and need DOM to be ready.

WidgetManager.init()

`init` simply initializes the `WidgetManager` and calls `init` methods of the related widgets if they define (see Widget JavaScript API section below)

```

```js
myWidgetManager.init();
```

```

WidgetManager.refresh()

`refresh` method refreshes all widgets related to this `WidgetManager`:

```

```
myWidgetManager.refresh();
```

```

WidgetManager Options

WidgetManager has some additional options.

Filter Form

If your widgets require parameters/filters then you will generally have a form to filter the widgets. In such cases, you can create a form that has some form elements and a dashboard area with some widgets inside. Example:

```
```xml
<form method="get" id="MyDashboardFilterForm">
 ... form elements
</form>

<div id="MyDashboardWidgetsArea" data-widget-filter="#MyDashboardFilterForm">
 ... widgets
</div>
```
```

`data-widget-filter` attribute relates the form with the widgets. Whenever the form is submitted, all the widgets are automatically refreshed with the form fields as the filter.

Instead of the `data-widget-filter` attribute, you can use the `filterForm` parameter of the `WidgetManager` constructor. Example:

```
```js
var myWidgetManager = new abp.WidgetManager({
 wrapper: '#MyDashboardWidgetsArea',
 filterForm: '#MyDashboardFilterForm'
});
```
```

Filter Callback

You may want to have a better control to provide filters while initializing and refreshing the widgets. In this case, you can use the `filterCallback` option:

```
```js
var myWidgetManager = new abp.WidgetManager({
 wrapper: '#MyDashboardWidgetsArea',
 filterCallback: function() {
 return $('#MyDashboardFilterForm').serializeFormToObject();
 }
});
```
```

This example shows the default implementation of the `filterCallback`. You can return any JavaScript object with fields. Example:

```
```js
filterCallback: function() {
 return {
 'startDate': $('#StartDateInput').val(),
```
```

```

        'endDate': $('#EndDateInput').val()
    };
}
```

```

The returning filters are passed to all widgets on `init` and `refresh`.

#### #### Widget JavaScript API

A widget can define a JavaScript API that is invoked by the `WidgetManager` when needed. The code sample below can be used to start to define an API for a widget.

```

```js
(function () {
    abp.widgets.NewUserStatisticWidget = function ($wrapper) {

        var getFilters = function () {
            return {
                ...
            };
        }

        var refresh = function (filters) {
            ...
        };

        var init = function (filters) {
            ...
        };

        return {
            getFilters: getFilters,
            init: init,
            refresh: refresh
        };
    };
})();
```

```

`NewUserStatisticWidget` is the name of the widget here. It should match the widget name defined in the server side. All of the functions are optional.

#### #### getFilters

If the widget has internal custom filters, this function should return the filter object. Example:

```

```js
var getFilters = function() {
    return {
        frequency: $wrapper.find('.frequency-filter option:selected').val()
    };
}
```

```

```
}\n}...
`
```

This method is used by the `WidgetManager` while building filters.

#### #### init

Used to initialize the widget when needed. It has a filter argument that can be used while getting data from server. `init` method is used when `WidgetManager.init()` function is called. It is also called if your widget requires a full re-load on refresh. See the `RefreshUrl` widget option.

#### #### refresh

Used to refresh the widget when needed. It has a filter argument that can be used while getting data from server. `refresh` method is used whenever `WidgetManager.refresh()` function is called.

### ## Authorization

Some widgets may need to be available only for authenticated or authorized users. In this case, use the following properties of the `Widget` attribute:

- \* `RequiresAuthentication` (`bool`): Set to true to make this widget usable only for authentication users (user have logged in to the application).
- \* `RequiredPolicies` (`List<string>`): A list of policy names to authorize the user. See [the authorization document](../../Authorization.md) for more info about policies.

Example:

```
```csharp\nusing Microsoft.AspNetCore.Mvc;\nusing Volo.Abp.AspNetCore.Mvc;\nusing Volo.Abp.AspNetCore.Mvc.UI.Widgets;\n\nnamespace DashboardDemo.Web.Pages.Components\n{\n    [Widget(RequiredPolicies = new[] { "MyPolicyName" })]\n    public class MySimpleWidgetViewComponent : AbpViewComponent\n    {\n        public IViewComponentResult Invoke()\n        {\n            return View();\n        }\n    }\n}...  
`
```

WidgetOptions

As alternative to the `Widget` attribute, you can use the `AbpWidgetOptions` to configure widgets:

```
```csharp
Configure<AbpWidgetOptions>(options =>
{
 options.Widgets.Add<MySimpleWidgetViewComponent>();
});
```

```

Write this into the `ConfigureServices` method of your [module](../../Module-Development-Basics.md). All the configuration done with the `Widget` attribute is also possible with the `AbpWidgetOptions`. Example configuration that adds a style for the widget:

```
```csharp
Configure<AbpWidgetOptions>(options =>
{
 options.Widgets
 .Add<MySimpleWidgetViewComponent>()
 .WithStyles("/Pages/Components/MySimpleWidget/Default.css");
});
```

```

> Tip: `AbpWidgetOptions` can also be used to get an existing widget and change its configuration. This is especially useful if you want to modify the configuration of a widget inside a module used by your application. Use `options.Widgets.Find` to get an existing `WidgetDefinition`.

See Also

- * [Example project (source code)](<https://github.com/abpframework/abp-samples/tree/master/DashboardDemo>).

10.1.14 Toolbars

ASP.NET Core MVC / Razor Pages UI: Toolbars

The Toolbar system is used to define ****toolbars**** on the user interface. Modules (or your application) can add ****items**** to a toolbar, then the [theme](Theming.md) renders the toolbar on the ****layout****.

There is only one ****standard toolbar**** named "Main" (defined as a constant: `StandardToolbars.Main`). The [Basic Theme](Basic-Theme) renders the main toolbar as shown below:

```
![bookstore-toolbar-highlighted](../../images/bookstore-toolbar-highlighted.png)
```

In the screenshot above, there are two items added to the main toolbar: Language switch component & user menu. You can add your own items here.

Also, [LeptonX Lite Theme](../../Themes/LeptonXLite/AspNetCore.md) has 2 different toolbars for desktop and mobile views which defined as constants: `LeptonXLiteToolbars.Main`, `LeptonXLiteToolbars.MainMobile`.

```
| LeptonXLiteToolbars.Main | LeptonXLiteToolbars.MainMobile |
| :---: | :---: |
| ! [leptonx] (../../images/leptonxlite-toolbar-main-example.png)
| ! [leptonx] (../../images/leptonxlite-toolbar-mainmobile-example.png) |
```

Example: Add a Notification Icon

In this example, we will add a **notification (bell) icon** to the left of the language switch item. A item in the toolbar should be a **view component**. So, first, create a new view component in your project:

```
![bookstore-notification-view-component] (../../images/bookstore-notification-view-component.png)
```

NotificationViewComponent.cs

```
~~~~csharp
public class NotificationViewComponent : AbpViewComponent
{
    public async Task<IViewComponentResult> InvokeAsync()
    {
        return View("/Pages/Shared/Components/Notification/Default.cshtml");
    }
}~~~~
```

Default.cshtml

```
~~~~xml
<div id="MainNotificationIcon" style="color: white; margin: 8px;">
    <i class="far fa-bell"></i>
</div>
~~~~
```

Now, we can create a class implementing the `IToolbarContributor` interface:

```
~~~~csharp
public class MyToolbarContributor : IToolbarContributor
{
    public Task ConfigureToolbarAsync(IToolbarConfigurationContext context)
    {
        if (context.Toolbar.Name == StandardToolbars.Main)
        {
            context.Toolbar.Items
                .Insert(0, new ToolbarItem(typeof(NotificationViewComponent)));
        }

        return Task.CompletedTask;
    }
}~~~~
```

You can use the [authorization](../../Authorization.md) to decide whether to add a `ToolbarItem`.

```
```csharp
if (await context.IsGrantedAsync("MyPermissionName"))
{
 //...add Toolbar items
}
````
```

You can use `RequirePermissions` extension method as a shortcut. It is also more performant, ABP optimizes the permission check for all the items.

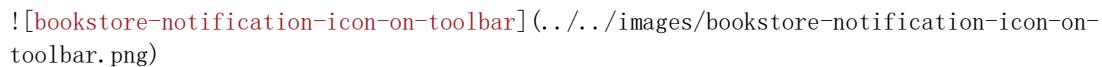
```
```csharp
context.Toolbar.Items.Insert(0, new
ToolbarItem(typeof(NotificationViewComponent)).RequirePermissions("MyPermissionName"));
````
```

This class adds the `NotificationViewComponent` as the first item in the `Main` toolbar.

Finally, you need to add this contributor to the `AbpToolbarOptions`, in the `ConfigureServices` of your [module](../../Module-Development-Basics.md):

```
```csharp
Configure<AbpToolbarOptions>(options =>
{
 options CONTRIBUTORS.Add(new MyToolbarContributor());
});
````
```

That's all, you will see the notification icon on the toolbar when you run the application:

A small screenshot of a user interface showing a toolbar with a notification icon.

`NotificationViewComponent` in this sample simply returns a view without any data. In real life, you probably want to **query database** (or call an HTTP API) to get notifications and pass to the view. If you need, you can add a `JavaScript` or `CSS` file to the global [bundle](Bundling-Minification.md) for your toolbar item.

IToolbarManager

`IToolbarManager` is used to render the toolbar. It returns the toolbar items by a toolbar name. This is generally used by the [themes](Theming.md) to render the toolbar on the layout.

```
# ASP.NET Core MVC / Razor Pages: Page Header
```

``IPageLayout`` service can be used to set the page title, selected menu item and the breadcrumb items for a page. It's the [`theme`](Theming.md)'s responsibility to render these on the page.

```
## IPageLayout
```

``IPageLayout`` can be injected in any page/view to set the page header properties.

```
### Page Title
```

Page Title can be set as shown in the example below:

```
```csharp
@inject IPageLayout PageLayout
 @{
 PageLayout.Content.Title = "Book List";
 }
```

```

* The Page Title is set to the HTML `title` tag (in addition to the [`brand/application name`](Branding.md)).

* The theme may render the Page Title before the Page Content (not implemented by the Basic Theme).

```
### Breadcrumb
```

```
> **The [Basic Theme](Basic-Theme.md) currently doesn't implement the breadcrumbs.**
>
> The [LeptonX Lite Theme](../../Themes/LeptonXLite/AspNetCore.md) supports breadcrumbs.
```

Breadcrumb items can be added to the `PageLayout.Content.BreadCrumb`.

****Example: Add Language Management to the breadcrumb items.****

```
```
PageLayout.Content.BreadCrumb.Add("Language Management");
```

```

The theme then renders the breadcrumb. An example render result can be:

```
![breadcrumbs-example](../../images/breadcrumbs-example.png)
```

* The Home icon is rendered by default. Set `PageLayout.Content.BreadCrumb.ShowHome` to `false` to hide it.
* Current Page name (got from the `PageLayout.Content.Title`) is added as the last by default. Set `PageLayout.Content.BreadCrumb.ShowCurrent` to `false` to hide it.

Any item that you add is inserted between Home and Current Page items. You can add as many item as you need. `BreadCrumb.Add(...)` method gets three parameters:

```
* `text`: The text to show for the breadcrumb item.  
* `url` (optional): A URL to navigate to, if the user clicks to the breadcrumb item.  
* `icon` (optional): An icon class (like `fas fa-user-tie` for Font-Awesome) to show with the `text`.
```

The Selected Menu Item

```
> **The [Basic Theme](Basic-Theme.md) currently doesn't implement the selected menu item  
since it is not applicable to the top menu which is the only option for the Basic Theme  
for now.**  
>  
> The [LeptonX Lite Theme](../../Themes/LeptonXLite/AspNetCore.md) supports selected menu  
item.
```

You can set the Menu Item name related to this page:

```
```csharp  
PageLayout.Content.MenuitemName = "BookStore.Books";
```
```

Menu item name should match a unique menu item name defined using the [Navigation / Menu](Navigation-Menu.md) system. In this case, it is expected from the theme to make the menu item "active" in the main menu.

10.1.16 Branding

ASP.NET Core MVC / Razor Pages: Branding

IBrandingProvider

`IBrandingProvider` is a simple interface that is used to show the application name and logo on the layout.

The screenshot below shows *MyProject* as the application name:

```
![branding-nobrand](../../images/branding-nobrand.png)
```

You can implement the `IBrandingProvider` interface or inherit from the `DefaultBrandingProvider` to set the application name:

```
```csharp  
using Volo.Abp.Ui.Branding;
using Volo.Abp.DependencyInjection;

namespace MyProject.Web
{
 [Dependency(ReplaceServices = true)]
 public class MyProjectBrandingProvider : DefaultBrandingProvider
 {
 public override string AppName => "Book Store";
 }
}
```

```
 public override string LogoUrl => "/logo.png";
 }
}
```

The result will be like shown below:

```
![bookstore-added-logo](../../../../images/bookstore-added-logo.png)
```

`IBrandingProvider` has the following properties:

- \* `AppName`: The application name.
- \* `LogoUrl`: A URL to show the application logo.
- \* `LogoReverseUrl`: A URL to show the application logo on a reverse color theme (dark, for example).

> **Tip**: `IBrandingProvider` is used in every page refresh. For a multi-tenant application, you can return a tenant specific application name to customize it per tenant.

#### ## Overriding the Branding Area

You can see the [UI Customization Guide] (Customization-User-Interface.md) to learn how you can replace the branding area with a custom view component.

### 10.1.17 Layout Hooks

#### # ASP.NET Core MVC / Razor Pages: Layout Hooks

ABP Framework theming system places the page layout into the [theme] (Theming.md) NuGet packages. That means the final application doesn't include a `Layout.cshtml`, so you can't directly change the layout code to customize it.

You copy the theme code into your solution. In this case you are completely free to customize it. However, then you won't be able to get automatic updates of the theme (by upgrading the theme NuGet package).

ABP Framework provides different ways of [customizing the UI] (Customization-User-Interface.md).

The **Layout Hook System** allows you to **add code** at some specific parts of the layout. All layouts of all themes should implement these hooks. Finally, you can add a **view component** into a hook point.

#### ## Example: Add Google Analytics Script

Assume that you need to add the Google Analytics script to the layout (that will be available for all the pages). First, **create a view component** in your project:

```
![bookstore-google-analytics-view-component](../../images/bookstore-google-analytics-view-component.png)
```

#### \*\*NotificationViewComponent.cs\*\*

```
```csharp
public class GoogleAnalyticsViewComponent : AbpViewComponent
{
    public IViewComponentResult Invoke()
    {
        return View("/Pages/Shared/Components/GoogleAnalytics/Default.cshtml");
    }
}
````
```

#### \*\*Default.cshtml\*\*

```
```html
<script>
    (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
        (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
        m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
    })(window,document,'script','//www.google-analytics.com/analytics.js','ga');

    ga('create', 'UA-xxxxxx-1', 'auto');
    ga('send', 'pageview');
</script>
````
```

Change `UA-xxxxxx-1` with your own code.

You can then add this component to any of the hook points in the `ConfigureServices` of your module:

```
```csharp
Configure<AbpLayoutHookOptions>(options =>
{
    options.Add(
        LayoutHooks.Head.Last, //The hook name
        typeof(GoogleAnalyticsViewComponent) //The component to add
    );
})
````
```

Now, the GA code will be inserted in the `head` of the page as the last item.

#### ### Specifying the Layout

The configuration above adds the `GoogleAnalyticsViewComponent` to all layouts. You may want to only add to a specific layout:

```

```csharp
Configure<AbpLayoutHookOptions>(options =>
{
    options.Add(
        LayoutHooks.Head.Last,
        typeof(GoogleAnalyticsViewComponent),
        layout: StandardLayouts.Application //Set the layout to add
    );
});
```

```

See the *\*Layouts\** section below to learn more about the layout system.

## ## Layout Hook Points

There are some pre-defined layout hook points. The `LayoutHooks.Head.Last` used above was one of them. The standard hook points are;

- \* `LayoutHooks.Head.First` : Used to add a component as the first item in the HTML head tag.
- \* `LayoutHooks.Head.Last` : Used to add a component as the last item in the HTML head tag.
- \* `LayoutHooks.Body.First` : Used to add a component as the first item in the HTML body tag.
- \* `LayoutHooks.Body.Last` : Used to add a component as the last item in the HTML body tag.
- \* `LayoutHooks.PageContent.First` : Used to add a component just before the page content (the `@RenderBody()` in the layout).
- \* `LayoutHooks.PageContent.Last` : Used to add a component just after the page content (the `@RenderBody()` in the layout).

> You (or the modules you are using) can add **multiple items to the same hook point**. All of them will be added to the layout by the order they were added.

## ## Layouts

Layout system allows themes to define standard, named layouts and allows any page to select a proper layout for its purpose. There are three pre-defined layouts:

- \* **Application**: The main (and the default) layout for an application. It typically contains header, menu (sidebar), footer, toolbar... etc.
- \* **Account**: This layout is used by login, register and other similar pages. It is used for the pages under the `/Pages/Account` folder by default.
- \* **Empty**: Empty and minimal layout.

These names are defined in the `StandardLayouts` class as constants. You can definitely create your own layouts, but these are the standard layout names and implemented by all the themes out of the box.

## ### Layout Location

You can find the layout files

[here](<https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic/Themes/Basic/Layouts>) for the basic

theme. You can take them as references to build your own layouts or you can override them if necessary.

## ## See Also

- \* [Customizing the User Interface](Customization-User-Interface.md)

### 10.1.18 Testing

#### # ASP.NET Core MVC / Razor Pages: Testing

> You can follow the [ASP.NET Core Integration Tests documentation](<https://docs.microsoft.com/en-us/aspnet/core/test/integration-tests>) to learn details of ASP.NET Core integration tests. This document explains the additional test infrastructure provided by the ABP Framework.

#### ## The Application Startup Template

The Application Startup Template contains the `.`.Web` project that contains UI views/pages/components of the application and a `.`.Web.Tests` project to test these.

![aspnetcore-web-tests-in-solution](../../images/aspnetcore-web-tests-in-solution.png)

#### ## Testing the Razor Pages

Assume that you've created a Razor Page, named `Issues.cshtml` with the following contents;

##### \*\*Issues.cshtml.cs\*\*

```
```csharp
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc.RazorPages;
using MyProject.Issues;

namespace MyProject.Web.Pages
{
    public class IssuesModel : PageModel
    {
        public List<IssueDto> Issues { get; set; }

        private readonly IIssueAppService _issueAppService;

        public IssuesModel(IIssueAppService issueAppService)
        {
            _issueAppService = issueAppService;
        }

        public async Task OnGetAsync()
        {
```

```

        Issues = await _issueAppService.GetListAsync();
    }
}
```

```

### \*\*Issues.cshtml\*\*

```

~~~~html
@page
@model MyProject.Web.Pages.IssuesModel
<h2>Issue List</h2>
<table id="IssueTable" class="table">
    <thead>
        <tr>
            <th>Issue</th>
            <th>Closed?</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var issue in Model.Issues)
        {
            <tr>
                <td>@issue.Title</td>
                <td>
                    @if (issue.IsClosed)
                    {
                        <span>Closed</span>
                    }
                    else
                    {
                        <span>Open</span>
                    }
                </td>
            </tr>
        }
    </tbody>
</table>
```

```

This page simply creates a table with the issues:

```
![issue-list](../../images/issue-list.png)
```

You can write a test class inside the `Web.Tests` project just like the example below:

```

~~~~csharp
using System.Threading.Tasks;
using HtmlAgilityPack;
using Shouldly;
using Xunit;

```

```

namespace MyProject.Pages
{
 public class Issues_Tests : MyProjectWebTestBase
 {
 [Fact]
 public async Task Should_Get_Table_Of_Issues()
 {
 // Act

 var response = await GetResponseAsStringAsync("/Issues");

 //Assert

 var htmlDocument = new HtmlDocument();
 htmlDocument.LoadHtml(response);

 var tableElement = htmlDocument.GetElementById("IssueTable");
 tableElement.ShouldNotBeNull();

 var trNodes = tableElement.SelectNodes("//tbody/tr");
 trNodes.Count.ShouldBeGreaterThan(0);
 }
 }
}
```

```

``GetResponseAsStringAsync`` is a shortcut method that comes from the base class that performs a HTTP GET request, checks if the resulting HTTP Status is ``200`` and returns the response as a ``string``.

> You can use the base ``Client`` object (of type ``HttpClient``) to perform any kind of request to the server and read the response yourself. ``GetResponseAsStringAsync`` is just a shortcut method.

This example uses the `[HtmlAgilityPack]` (<https://html-agility-pack.net/>) library to parse the incoming HTML and test if it contains the issue table.

> This example assumes there are some initial issues in the database. See the **The Data Seed** section of the `[Testing document](../../Testing.md)` to learn how to seed test data, so your tests can assume some initial data available in the database.

Testing the Controllers

Testing a controller is not different. Just perform a request to the server with a proper URL, get the response and make your assertions.

View Result

If the controller returns a View, you can use a similar code to test the returned HTML. See the Razor Pages example above.

Object Result

If the controller returns an object result, you can use the `GetResponseAsObjectAsync` base method.

Assume that you've a controller as defined below:

```
```csharp
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using MyProject.Issues;
using Volo.Abp.AspNetCore.Mvc;

namespace MyProject.Web.Controllers
{
 [Route("api/issues")]
 public class IssueController : AbpController
 {
 private readonly IIssueAppService _issueAppService;

 public IssueController(IIssueAppService issueAppService)
 {
 _issueAppService = issueAppService;
 }

 [HttpGet]
 public async Task<List<IssueDto>> GetAsync()
 {
 return await _issueAppService.GetListAsync();
 }
 }
}
````
```

You can write a test code to execute the API and get the result:

```
```csharp
using System.Collections.Generic;
using System.Threading.Tasks;
using MyProject.Issues;
using Shouldly;
using Xunit;

namespace MyProject.Pages
{
 public class Issues_Tests : MyProjectWebTestBase
 {
 [Fact]
 public async Task Should_Get_Issues_From_Api()
 {
 var issues = await GetResponseAsObjectAsync<List<IssueDto>>("/api/issues");
 }
 }
}
````
```

```
        issues.ShouldNotBeNull();
        issues.Count.ShouldBeGreaterThan(0);
    }
}
```

```

## ## Testing the JavaScript Code

ABP Framework doesn't provide any infrastructure to test your JavaScript code. You can use any test framework and tooling to test your JavaScript code.

## ## The Test Infrastructure

[[Volo.Abp.AspNetCore.TestBase](#)] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.TestBase>) package provides the test infrastructure that is integrated to the ABP Framework and ASP.NET Core.

➤ Volo.Abp.AspNetCore.TestBase package is already installed in the ` `.Web.Tests` project.

This package provides the ` `AbpAspNetCoreIntegratedTestBase` as the fundamental base class to derive the test classes from. The ` `MyProjectWebTestBase` base class used above inherits from the ` `AbpAspNetCoreIntegratedTestBase` , so we indirectly inherited the ` `AbpAspNetCoreIntegratedTestBase` .

### ### Base Properties

The ` `AbpAspNetCoreIntegratedTestBase` provides the following base properties those are used in the tests:

- \* ` `Server` : A ` `TestServer` instance that hosts the web application in tests.
- \* ` `Client` : An ` `HttpClient` instance that is configured to perform requests to the test server.
- \* ` `ServiceProvider` : The service provider that you can resolve services in case of need.

### ### Base Methods

` `AbpAspNetCoreIntegratedTestBase` provides the following methods that you can override if you need to customize the test server:

- \* ` `ConfigureServices` : can be overridden to register/replace services only for the derived test class.
- \* ` `CreateHostBuilder` : can be used to customize building the ` `IHostBuilder` .

See Also

\* [Overall / Server Side Testing] (./../Testing.md)

## 10.1.19 Theming

# ASP.NET Core MVC / Razor Pages: UI Theming

## ## Introduction

ABP Framework provides a complete **UI Theming** system with the following goals:

- \* Reusable [application modules](../../Modules/Index.md) are developed **theme-independent**, so they can work with any UI theme.
- \* UI theme is **decided by the final application**.
- \* The theme is distributed via NuGet/NPM packages, so it is **easily upgradable**.
- \* The final application can **customize** the selected theme.

In order to accomplish these goals, ABP Framework;

- \* Determines a set of **base libraries** used and adapted by all the themes. So, module and application developers can depend on and use these libraries without depending on a particular theme.
- \* Provides a system that consists of [navigation menus](Navigation-Menu.md), [toolbars](Toolbars.md), [layout hooks](Layout-Hooks.md)... that is implemented by all the themes. So, the modules and the application to contribute to the layout to compose a consistent application UI.

### ### Current Themes

Currently, four themes are **officially provided**:

- \* The [Basic Theme](Basic-Theme.md) is the minimalist theme with the plain Bootstrap style. It is **open source and free**.
- \* The [LeptonX Lite Theme](../../Themes/LeptonXLite/AspNetCore.md) is modern and stylish Bootstrap UI theme. It is ideal if you want to have a production ready UI theme. It is also **open source and free**.
- \* The [Lepton Theme](<https://commercial.abp.io/themes>) is a **commercial** theme developed by the core ABP team and is a part of the [ABP Commercial](<https://commercial.abp.io/>) license.
- \* The [LeptonX Theme](<https://docs.abp.io/en/commercial/latest/themes/lepton-x/index>) is also a **commercial** theme developed by the core ABP theme and is a part of the [ABP Commercial](<https://commercial.abp.io/>) license. This is the default theme after ABP v6.0.0.

There are also some community-driven themes for the ABP Framework (you can search on the web).

## ## Overall

### ### The Base Libraries

All the themes must depend on the [abp/aspnetcore.mvc.ui.theme.shared](<https://www.npmjs.com/package/@abp/aspnetcore.mvc.ui.theme.shared>) NPM package, so they are indirectly depending on the following libraries:

- \* [Twitter Bootstrap](<https://getbootstrap.com/>) as the fundamental HTML/CSS framework.
- \* [JQuery](<https://jquery.com/>) for DOM manipulation.
- \* [DataTables.Net](<https://datatables.net/>) for data grids.

- \* [JQuery Validation](https://github.com/jquery-validation/jquery-validation) for client side & [unobtrusive](https://github.com/aspnet/jquery-validation-unobtrusive) validation
- \* [FontAwesome](https://fontawesome.com/) as the fundamental CSS font library.
- \* [SweetAlert](https://sweetalert.js.org/) to show fancy alert message and confirmation dialogs.
- \* [ Toastr](https://github.com/CodeSeven/toastr) to show toast notifications.
- \* [Lodash](https://lodash.com/) as a utility library.
- \* [Luxon](https://moment.github.io/luxon/) for date/time operations.
- \* [JQuery Form](https://github.com/jquery-form/form) for AJAX forms.
- \* [bootstrap-datepicker](https://github.com/uxsolutions/bootstrap-datepicker) to show date pickers.
- \* [Select2](https://select2.org/) for better select/combo boxes.
- \* [Timeago](http://timeago.yarp.com/) to show automatically updating fuzzy timestamps.
- \* [malihu-custom-scrollbar-plugin](https://github.com/malihu/malihu-custom-scrollbar-plugin) for custom scrollbars.

These libraries are selected as the base libraries and available to the applications and modules.

#### #### Abstractions / Wrappers

There are some abstractions in the ABP Framework to make your code independent from some of these libraries too. Examples;

- \* [Tag Helpers](Tag-Helpers/Index.md) makes it easy to generate the Bootstrap UIs.
- \* JavaScript [Message](JavaScript-API/Message.md) and [Notification](JavaScript-API/Notify.md) APIs provides abstractions to use the Sweetalert and Toastr.
- \* [Forms & Validation](Forms-Validation.md) system automatically handles the validation, so you mostly don't directly type any validation code.

#### ### The Standard Layouts

The main responsibility of a theme is to provide the layouts. There are **\*\*three predefined layouts must be implemented by all the themes\*\***:

- \* **\*\*Application\*\***: The default layout which is used by the main application pages.
- \* **\*\*Account\*\***: Mostly used by the [account module](../../Modules/Account.md) for login, register, forgot password... pages.
- \* **\*\*Empty\*\***: The Minimal layout that has no layout components at all.

Layout names are constants defined in the ``Volo.Abp.AspNetCore.Mvc.UI.Theming.StandardLayouts`` class.

#### #### The Application Layout

This is the default layout which is used by the main application pages. The following image shows the user management page in the [Basic Theme](Basic-Theme.md) application layout:

`![basic-theme-application-layout](../../images/basic-theme-application-layout.png)`

And the same page is shown below with the [Lepton Theme] (<https://commercial.abp.io/themes>) application layout:

![lepton-theme-application-layout](../../images/lepton-theme-application-layout.png)

As you can see, the page is the same, but the look is completely different in the themes above.

The application layout typically includes the following parts;

- \* A [main menu] (Navigation-Menu.md)
- \* Main [Toolbar] (Toolbars.md) with the following components;
  - \* User menu
  - \* Language switch dropdown
- \* [Page alerts] (Page-Alerts.md)
- \* The page content (aka `RenderBody()`)
- \* [Layout hooks] (Layout-Hooks.md)

Some themes may provide more parts like breadcrumbs, page header & toolbar... etc. See the *\*Layout Parts\** section.

#### #### The Account Layout

The Account layout is typically used by the [account module] (../../Modules/Account.md) for login, register, forgot password... pages.

![basic-theme-account-layout](../../images/basic-theme-account-layout.png)

This layout typically provides the following parts;

- \* Language switch dropdown
- \* Tenant switch area (if the application is [multi-tenant] (../../Multi-Tenancy.md) and the current is resolved by the cookie)
- \* [Page alerts] (Page-Alerts.md)
- \* The page content (aka `RenderBody()`)
- \* [Layout hooks] (Layout-Hooks.md)

The [Basic Theme] (Basic-Theme.md) renders the top navigation bar for this layout too (as shown above)

Here, the account layout of the Lepton Theme:

![lepton-theme-account-layout](../../images/lepton-theme-account-layout.png)

The [Lepton Theme] (<https://commercial.abp.io/themes>) shows the application logo and footer in this layout.

➤ You can override theme layouts completely or partially in an application to [customize] (Customization-User-Interface.md) it.

#### #### The Empty Layout

The empty layout provides an empty page. It typically includes the following parts;

- \* [Page alerts] (Page-Alerts.md)
- \* The page content (aka `RenderBody()`)
- \* [Layout hooks] (Layout-Hooks.md)

## ## Implementing a Theme

### ### The Easiest Way

The easiest way of creating a new theme is adding [Basic Theme Source Code] (<https://github.com/abpframework/abp/tree/dev/modules/basic-theme>) module with source codes and customizing it.

```
```bash
abp add-package Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic --with-source-code --add-to-solution-file
````
```

### ### The ITheme Interface

`ITheme` interface is used by the ABP Framework to select the layout for the current page. A theme must implement this interface to provide the requested layout path.

This is the `ITheme` implementation of the [Basic Theme] (Basic-Theme.md).

```
```csharp
using Volo.Abp.AspNetCore.Mvc.UI.Theming;
using Volo.Abp.DependencyInjection;

namespace Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic
{
    [ThemeName("Name")]
    public class BasicTheme : ITheme, ITransientDependency
    {
        public const string Name = "Basic";

        public virtual string GetLayout(string name, bool fallbackToDefault = true)
        {
            switch (name)
            {
                case StandardLayouts.Application:
                    return "~/Themes/Basic/Layouts/Application.cshtml";
                case StandardLayouts.Account:
                    return "~/Themes/Basic/Layouts/Account.cshtml";
                case StandardLayouts.Empty:
                    return "~/Themes/Basic/Layouts/Empty.cshtml";
                default:
                    return fallbackToDefault
                        ? "~/Themes/Basic/Layouts/Application.cshtml"
                        : null;
            }
        }
    }
}
```

```
        }
    }
}...  
``
```

- * `*[ThemeName]*` attribute is required and a theme must have a unique name, `Basic` in this sample.
- * `GetLayout` method should return a path if the requested layout (`name`) is provided by the theme. *The Standard Layouts* should be implemented if the theme is aimed to be used by a standard application. It may implement additional layouts.

Once the theme implements the `ITheme` interface, it should add the theme to the `AbpThemingOptions` in the `ConfigureServices` method of the [module](`Module-Development-Basics.md`).

```
```csharp
Configure<AbpThemingOptions>(options =>
{
 options.Themes.Add<BasicTheme>();
});
````
```

The IThemeSelector Service

ABP Framework allows to use multiple themes together. This is why `options.Themes` is a list. `IThemeSelector` service selects the theme on the runtime. The application developer can set the `AbpThemingOptions.DefaultThemeName` to set the theme to be used, or replace the `IThemeSelector` service implementation (the default implementation is `DefaultThemeSelector`) to completely control the theme selection on runtime.

Bundles

[Bundling system](Bundling-Minification.md) provides a standard way to import style & script files into pages. There are two standard bundles defined by the ABP Framework:

- * `StandardBundles.Styles.Global` : The global bundle that includes the style files used in all the pages. Typically, it includes the CSS files of the Base Libraries.
- * `StandardBundles.Scripts.Global` : The global bundle that includes the script files used in all the pages. Typically, it includes the JavaScript files of the Base Libraries.

A theme generally extends these standard bundles by adding theme specific CSS/JavaScript files.

The best way to define new bundles, inherit from the standard bundles and add to the `AbpBundlingOptions` as shown below (this code is from the [Basic Theme](Basic-Theme.md)):

```
```csharp
Configure<AbpBundlingOptions>(options =>
{
 options
 .StyleBundles
 .Add(BasicThemeBundles.Styles.Global, bundle =>
```

```

 {
 bundle
 .AddBaseBundles(StandardBundles.Styles.Global)
 .AddContributors(typeof(BasicThemeGlobalStyleContributor));
 });

 options
 .ScriptBundles
 .Add(BasicThemeBundles.Scripts.Global, bundle =>
 {
 bundle
 .AddBaseBundles(StandardBundles.Scripts.Global)
 .AddContributors(typeof(BasicThemeGlobalScriptContributor));
 });
}
```

```

``BasicThemeGlobalStyleContributor`` and ``BasicThemeGlobalScriptContributor`` are bundle contributors. For example, ``BasicThemeGlobalStyleContributor`` is defined as shown below:

```

```csharp
public class BasicThemeGlobalStyleContributor : BundleContributor
{
 public override void ConfigureBundle(BundleConfigurationContext context)
 {
 context.Files.Add("/themes/basic/layout.css");
 }
}
```

```

Then the theme can render these bundles in a layout. For example, you can render the Global Styles as shown below:

```

```html
<abp-style-bundle name="@BasicThemeBundles.Styles.Global" />
```

```

See the [Bundle & Minification] (Bundling-Minification.md) document to understand the Bundling system better.

Layout Parts

A typical Layout consists of several parts. The theme should include the necessary parts in each layout.

****Example: The Basic Theme has the following parts for the Application Layout****

```

![basic-theme-application-layout-parts](../../images/basic-theme-application-layout-parts.png)

```

The application code and the modules can only show contents in the Page Content part. If they need to change the other parts (to add a menu item, to add a toolbar item, to change the application name in the branding area...) they should use the ABP Framework APIs.

The following sections explain the fundamental parts pre-defined by the ABP Framework and can be implemented by the themes.

- It is a good practice to split the layout into components/partials, so the final application can override them partially for customization purpose.

Branding

`IBrandingProvider` service should be used to get the name and the logo URL of the application to render in the Branding part.

The [Application Startup Template](../../Startup-Templates/Application.md) has an implementation of this interface to set the values by the application developer.

Main Menu

`IMenuManager` service is used to get the main menu items and render on the layout.

****Example: Get the Main Menu to render in a view component****

```
```csharp
public class MainNavbarMenuViewComponent : AbpViewComponent
{
 private readonly IMenuManager _menuManager;

 public MainNavbarMenuViewComponent(IMenuManager menuManager)
 {
 _menuManager = menuManager;
 }

 public async Task<IViewComponentResult> InvokeAsync()
 {
 var menu = await _menuManager.GetAsync(StandardMenus.Main);
 return View("~/Themes/Basic/Components/Menu/Default.cshtml", menu);
 }
}
```

```

See the [Navigation / Menus](Navigation-Menu.md) document to learn more about the navigation system.

Main Toolbar

`IToolbarManager` service is used to get the Main Toolbar items and render on the layout. Each item of this toolbar is a View Component, so it may include any type of UI elements. Inject the `IToolbarManager` and use the `GetAsync` to get the toolbar items:

```
```csharp

```

```
var toolbar = await _toolbarManager.GetAsync(StandardToolbars.Main);
```
```

➤ See the [Toolbars](Toolbars.md) document to learn more on the toolbar system.

The theme has a responsibility to add two pre-defined items to the main toolbar: Language Selection and User Menu. To do that, create a class implementing the `IToolbarContributor` interface and add it to the `AbpToolbarOptions` as shown below:

```
```csharp  
Configure<AbpToolbarOptions>(options =>
{
 options CONTRIBUTORS.Add(new BasicThemeMainTopToolbarContributor());
});
```
```

Language Selection

Language Selection toolbar item is generally a dropdown that is used to switch between languages. `ILanguageProvider` is used to get the list of available languages and `CultureInfo.CurrentUICulture` is used to learn the current language.

`/Abp/Languages/Switch` endpoint can be used to switch the language. This endpoint accepts the following query string parameters:

- * `culture`: The selected culture, like `en-US` or `en`.
- * `uiCulture`: The selected UI culture, like `en-US` or `en`.
- * `returnUrl` (optional): Can be used to return a given URL after switching the language.

`culture` and `uiCulture` should match one of the available languages. ABP Framework sets a culture cookie in the `/Abp/Languages/Switch` endpoint.

User Menu

User menu includes links related to the user account. `IMenuManager` is used just like the Main Menu, but this time with `StandardMenus.User` parameter like shown below:

```
```csharp  
var menu = await _menuManager.GetAsync(StandardMenus.User);
```
```

[ICurrentUser](../../CurrentUser.md) and [ICurrentTenant](../../Multi-Tenancy.md) services can be used to obtain the current user and tenant names.

Page Alerts

`IAlertManager` service is used to get the current page alerts to render on the layout. Use the `Alerts` list of the `IAlertManager`. It is generally rendered just before the page content (`RenderBody()`).

See the [Page Alerts](Page-Alerts.md) document to learn more.

Layout Hooks

Since the Layout is in the theme package, the final application or any module can't directly manipulate the layout content. The [Layout Hook](Layout-Hooks.md) system allows to inject components to some specific points of the layout.

The theme is responsible to render the hooks in the correct place.

****Example: Render the `LayoutHooks.Head.First` Hook in the Application Layout****

```
```html
<head>
 @await Component.InvokeLayoutHookAsync(LayoutHooks.Head.First,
StandardLayouts.Application)
 ...
```
```

See the [Layout Hook](Layout-Hooks.md) document to learn the standard layout hooks.

Script / Style Sections

Every layout should render the following optional sections:

- * `styles` section is rendered in the end of the `head`, just before the `LayoutHooks.Head.Last`.
- * `scripts` section is rendered in the end of the `body`, just before the `LayoutHooks.Body.Last`.

In this way, the page can import styles and scripts to the layout.

****Example: Render the `styles` section****

```
```csharp
@await RenderSectionAsync("styles", required: false)
```
```

Content Toolbar Section

Another pre-defined section is the Content Toolbar section which can be used by the pages to add code just before the page content. The Basic Theme renders it as shown below:

```
```html
<div id="AbpContentToolbar">
 <div class="text-end mb-2">
 @RenderSection("content_toolbar", false)
 </div>
</div>
```
```

The container div's id must be `AbpContentToolbar`. This section should come before the `RenderBody()`.

Widget Resources

The [Widget System](Widgets.md) allows to define reusable widgets with their own style/script files. All the layouts should render the widget style and scripts.

Widget Styles is rendered as shown below, just before the `styles` section, after the global style bundle:

```
```csharp
@await Component.InvokeAsync(typeof(WidgetStylesViewComponent))
```
```

Widget Scripts is rendered as shown below, just before the `scripts` section, after the global script bundle:

```
```csharp
@await Component.InvokeAsync(typeof(WidgetScriptsViewComponent))
```
```

ABP Scripts

ABP has some special scripts those should be included into every layout. They are not included in the global bundles since they are dynamically created based on the current user.

ABP scripts (`ApplicationConfigurationScript` and `ServiceProxyScript`) should be added just after the global script bundle, as shown below:

```
```html
<script src="~/Abp/ApplicationConfigurationScript"></script>
<script src="~/Abp/ServiceProxyScript"></script>
```
```

Page Title, Selected Menu Item and Breadcrumbs

`IPageLayout` service can be injected by any page to set the Page Title, the selected menu item name and the breadcrumb items. Then the theme can use this service to get these values and render on the UI.

The Basic Theme doesn't implement this service, but the Lepton Theme implements:

```
![breadcrumbs-example](../../images/breadcrumbs-example.png)
```

See the [Page Header](Page-Header.md) document for more.

Tenant Switch

The Account Layout should allow the user to switch the current tenant if the application is multi-tenant and the tenant was resolved from the cookies. See the [Basic Theme Account Layout](https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic/Themes/Basic/Layouts/Account.cshtml) as an example implementation.

Layout Classes

The Standard Layouts (`Application`, `Account` and `Empty`) should add the following CSS classes to the `body` tag:

- * `abp-application-layout` for the `Application` layout.
- * `abp-account-layout` for the `Account` layout.
- * `abp-empty-layout` for the `Empty` layout.

In this way, applications or modules can have selectors based on the current layout.

RTL

To support Right-To-Left languages, the Layout should check the current culture and add `dir="rtl"` to the `html` tag and `rtl` CSS class the the `body` tag.

You can check `CultureInfo.CurrentCulture.TextInfo.IsRightToLeft` to understand if the current language is a RTL language.

The NPM Package

A theme should have a NPM package that depends on the [@abp/aspnetcore.mvc.ui.theme.shared](https://www.npmjs.com/package/@abp/aspnetcore.mvc.ui.theme.shared) package. In this way, it inherits all the Base Libraries. If the theme requires additional libraries, then it should define these dependencies too.

Applications use the [Client Side Package Management](#) (Client-Side-Package-Management.md) system to add client side libraries to the project. So, if an application uses your theme, it should add dependency to your theme's NPM package as well as the NuGet package dependency.

10.1.19.1 The Basic Theme

ASP.NET Core MVC / Razor Pages: The Basic Theme

The Basic Theme is a theme implementation for the ASP.NET Core MVC / Razor Pages UI. It is a minimalist theme that doesn't add any styling on top of the plain [Bootstrap](https://getbootstrap.com/). You can take the Basic Theme as the **base theme** and build your own theme or styling on top of it. See the *Customization* section.

The Basic Theme has RTL (Right-to-Left language) support.

- If you are looking for a professional, enterprise ready theme, you can check the [Lepton Theme](https://commercial.abp.io/themes), which is a part of the [ABP Commercial](https://commercial.abp.io/).
- See the [Theming document](#) (Theming.md) to learn about themes.

Installation

If you need to manually this theme, follow the steps below:

- * Install the [\[Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic\]](https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic) (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic>) NuGet package to your web project.
- * Add `AbpAspNetCoreMvcUiBasicThemeModule` into the `[DependsOn(...)]` attribute for your [module class] (./Module-Development-Basics.md) in the web project.
- * Install the [@abp/aspnetcore.mvc.ui.theme.basic](https://www.npmjs.com/package/@abp/aspnetcore.mvc.ui.theme.basic) (<https://www.npmjs.com/package/@abp/aspnetcore.mvc.ui.theme.basic>) NPM package to your web project (e.g. `npm install @abp/aspnetcore.mvc.ui.theme.basic` or `yarn add @abp/aspnetcore.mvc.ui.theme.basic`).
- * Run `abp install-libs` command in a command line terminal in the web project's folder.

Layouts

The Basic Theme implements the standard layouts. All the layouts implement the following parts;

- * Global [\[Bundles\]](#) (Bundling-Minification.md)
- * [\[Page Alerts\]](#) (Page-Alerts.md)
- * [\[Layout Hooks\]](#) (Layout-Hooks.md)
- * [\[Widget\]](#) (Widgets.md) Resources

The Application Layout

![basic-theme-application-layout](./../images/basic-theme-application-layout.png)

Application Layout implements the following parts, in addition to the common parts mentioned above;

- * Branding
- * Main [\[Menu\]](#) (Navigation-Menu.md)
- * Main [\[Toolbar\]](#) (Toolbars.md) with Language Selection & User Menu

The Account Layout

![basic-theme-account-layout](./../images/basic-theme-account-layout.png)

Application Layout implements the following parts, in addition to the common parts mentioned above;

- * Branding
- * Main [\[Menu\]](#) (Navigation-Menu.md)
- * Main [\[Toolbar\]](#) (Toolbars.md) with Language Selection & User Menu
- * Tenant Switch Area

Empty Layout

Empty layout is empty, as its name stands for. However, it implements the common parts mentioned above.

Customization

You have two options two customize this theme:

Overriding Styles/Components

In this approach, you continue to use the the theme as NuGet and NPM packages and customize the parts you need to. There are several ways to customize it;

Override the Styles

1. Create a CSS file in the `wwwroot` folder of your project:

```
![example-global-styles](../../images/example-global-styles.png)
```

2. Add the style file to the global bundle, in the `ConfigureServices` method of your [module](../../Module-Development-Basics.md):

```
```csharp
Configure<AbpBundlingOptions>(options =>
{
 options.StyleBundles.Configure(BasicThemeBundles.Styles.Global, bundle =>
 {
 bundle.AddFiles("/styles/global-styles.css");
 });
});
```

#### #### Override the Components

See the [User Interface Customization Guide](Customization-User-Interface.md) to learn how you can replace components, customize and extend the user interface.

### ### Copy & Customize

You can run the following [ABP CLI](../../CLI.md) command in \*\*Web\*\* project directory to copy the source code to your solution:

```
`abp add-module Volo.BasicTheme --with-source-code --add-to-solution-file`
```

----

Or, you can download the [source code](https://github.com/abpframework/abp/tree/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic) of the Basic Theme, manually copy the project content into your solution, re-arrange the package/module dependencies (see the Installation section above to understand how it was installed to the project) and freely customize the theme based on your application requirements.

## ## See Also

\* [Theming] (Theming.md)

### 10.1.19.2 LeptonX Lite

#### # LeptonX Lite MVC UI

LeptonX Lite has implementation for the ABP Framework Razor Pages. It's a simplified variation of the [LeptonX Theme] (<https://x.leptontheme.com/>).

- > If you are looking for a professional, enterprise ready theme, you can check the [LeptonX Theme] (<https://x.leptontheme.com/>), which is a part of [ABP Commercial] (<https://commercial.abp.io/>).
- > See the [Theming document] (<https://docs.abp.io/en/abp/latest/UI/AspNetCore/Theming>) to learn about themes.

#### ## Installation

This theme is **\*\*already installed\*\*** when you create a new solution using the startup templates. If you are using any other template, you can install this theme by following the steps below:

- Add the **\*\*Volo.Abp.AspNetCore.Mvc.UI.Theme.LeptonXLite\*\*** package to your **\*\*Web\*\*** application.

```
```bash
dotnet add package Volo.Abp.AspNetCore.Mvc.UI.Theme.LeptonXLite --prerelease
````
```

- Remove the **\*\*Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic\*\*** reference from the project since it's not necessary after switching to LeptonX Lite.
- Make sure the old theme is removed and LeptonX is added in your Module class.

```
```diff
[DependsOn(
    // Remove the BasicTheme module from DependsOn attribute
    - typeof(AbpAspNetCoreMvcUiBasicThemeModule),
    // Add the LeptonX Lite module to DependsOn attribute
    + typeof(AbpAspNetCoreMvcUiLeptonXLiteThemeModule),
)]
````
```

- Replace `BasicThemeBundles` with `LeptonXLiteThemeBundles` in `AbpBundlingOptions`:

```
```diff
Configure<AbpBundlingOptions>(options =>
{
    options.StyleBundles.Configure(
        // Remove the following line
````
```

```

- BasicThemeBundles.Styles.Global,
// Add the following line instead
+ LeptonXLiteThemeBundles.Styles.Global,
bundle =>
{
 bundle.AddFiles("/global-styles.css");
}
);
```
## Customization

#### Layouts
```

LeptonX Lite MVC provides **layouts** for your **user interface** based [ABP Framework Theming](<https://docs.abp.io/en/abp/latest/UI/AspNetCore/Theming>). You can use **layouts** to **organize your user interface**.

The main responsibility of a theme is to **provide** the layouts. There are **three pre-defined layouts that must be implemented by all the themes:**

- * **Application:** The **default** layout which is used by the **main** application pages.
- * **Account:** Mostly used by the **account module** for **login**, **register**, **forgot password**... pages.
- * **Empty:** The **Minimal** layout that **has no layout components** at all.

Layout names are **constants** defined in the `LeptonXLiteTheme` class in the **MVC** project **root**.

> The layout pages define under the `Themes/LeptonXLite/Layouts` folder and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Toolbars

LeptonX Lite includes separated toolbars for desktop & mobile. You can manage toolbars independently. Toolbar names can be accessible in the **LeptonXLiteToolbars** class.

```

- `LeptonXLiteToolbars.Main`
- `LeptonXLiteToolbars.MainMobile`


```csharp
public class MyProjectNameMainToolbarContributor : IToolbarContributor
{
 public async Task ConfigureToolbarAsync(IToolbarConfigurationContext context)
 {
 if (context.Toolbar.Name == LeptonXLiteToolbars.Main)
 {
 context.Toolbar.Items.Add(new ToolbarItem(typeof(MyDesktopComponent)));
 }
 }
}
```

```

 }

 if (context.Toolbar.Name == LeptonXLiteToolbars.MainMobile)
 {
 context.Toolbar.Items.Add(new ToolbarItem(typeof(MyMobileComponent)));
 }
 }
}
```

```

LeptonX Lite MVC Components

ABP **helps** you make **highly customizable UI**. You can easily **customize** your themes to fit your needs. **The Virtual File System** makes it possible to **manage files** that **do not physically** exist on the **file system** (disk). It's mainly used to embed **(js, css, image..)** files into assemblies and **use them like** physical files at runtime. An application (or another module) can **override** a **virtual file of a module** just like placing a file with the **same name** and **extension** into the **same folder** of the **virtual file**.

LeptonX Lite is built on the [Abp Framework] (<https://abp.io/>), so you can **easily** customize your Asp. Net Core MVC user interface by following [Abp Mvc UI Customization] (<https://docs.abp.io/en/abp/latest/UI/AspNetCore/Customization-xUser-Interface>).

Branding Component

The **branding component** is a simple component that can be used to display your brand. It contains a **logo** and a **company name**.

![Brand component] (../../images/leptonxlite-brand-component.png)

How to override the Branding Component in LeptonX Lite MVC

* The **branding component page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/Brand/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **branding component (C# file)** is defined in the `Themes/LeptonXLite/Components/Brand/MainNavbarBrandViewComponent.cs` file and you can **override it** by creating a file with the **same name** and under the **same folder**.

How to override the favicon in LeptonX Lite MVC

You can add a new favicon to the `~/wwwroot/favicon.svg` and `~/wwwroot/favicon.ico` paths to override the current favicon.

Breadcrumb Component

On websites that have a lot of pages, **breadcrumb navigation** can greatly **enhance the way users find their way** around. In terms of **usability**, breadcrumbs reduce the

number of actions a website **visitor** needs to take in order to get to a **higher-level page**, and they **improve** the **findability** of **website sections** and **pages**.

![Breadcrumb component](../../images/leptonxlite-breadcrumb-component.png)

How to override the Breadcrumb Component in LeptonX Lite MVC

* The **breadcrumb component page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/Breadcrumbs/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **breadcrumb component (C# file)** is defined in the `Themes/LeptonXLite/Components/Breadcrumbs/BreadcrumbsViewComponent.cs` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Sidebar Menu Component

Sidebar menus have been used as **a directory for Related Pages** to a **Service** offering, **Navigation** items to a **specific service** or topic and even just as **Links** the user may be interested in.

![Sidebar menu component](../../images/leptonxlite-sidebar-menu-component.png)

How to override the Sidebar Menu Component in LeptonX Lite MVC

* **Sidebar menu page (.cshtml)** is defined in the `Themes/LeptonXLite/Components/Menu/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* If you want to **override the menu component (C#)** you can override the `Themes/LeptonXLite/Components/Menu/MainMenuItem.cs` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

> The **sidebar menu** renders menu items **dynamically**. The **menu item** is a **partial view** and is defined in the `Themes/LeptonXLite/Components/Menu/_MenuItem.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Page Alerts Component

Provides contextual **feedback messages** for typical user actions with the handful of **available** and **flexible** **alert messages**. Alerts are available for any length of text, as well as an **optional dismiss button**.

![Page alerts component](../../images/leptonxlite-page-alerts-component.png)

How to override the Page Alerts Component in LeptonX Lite MVC

* The **page alerts component page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/PageAlerts/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **page alerts component (C#)** is defined in the ``Themes/LeptonXLite/Components/PageAlerts/PageAlertsViewComponent.cs`` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Toolbar Component

Toolbar items are used to add **extra functionality to the toolbar**. The toolbar is a **horizontal bar** that **contains** a group of **toolbar items**.

How to override the Toolbar Component in LeptonX Lite MVC

* The **toolbar component page (.cshtml file)** is defined in the ``Themes/LeptonXLite/Components/Toolbar/Default.cshtml`` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **toolbar component (C#)** is defined in the ``Themes/LeptonXLite/Components/Toolbar/ToolbarViewComponent.cs`` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Toolbar Item Component

The toolbar item is a **single item** that **contains** a **link**, an **icon**, a **label** etc..

How to override the Toolbar Item Component in LeptonX Lite MVC

* The **toolbar item component page (.cshtml file)** is defined in the ``Themes/LeptonXLite/Components/ToolbarItems/Default.cshtml`` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **toolbar item component (C#)** is defined in the ``Themes/LeptonXLite/Components/ToolbarItems/ToolbarItemsViewComponent.cs`` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

You can find the toolbar components below:

Language Switch Component

Think about a **multi-lingual** website and the first thing that could **hit your mind** is **the language switch component**. A **navigation bar** is a **great place** to **embed a language switch**. By embedding the language switch in the navigation bar of your website, you would **make it simpler** for users to **find it** and **easily** switch the **language** `<u>`**without trying to locate it across the website.**`</u>`

![Language switch component](../../images/leptonxlite-language-switch-component.png)

How to override the Language Switch Component in LeptonX Lite MVC

* The **language switch component page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/LanguageSwitch/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **language switch component (C#)** is defined in the `Themes/LeptonXLite/Components/LanguageSwitch/LanguageSwitchViewComponent.cs` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Mobile Language Switch Component

The **mobile** **language switch component** is used to switch the language of the website **on mobile devices**. The mobile language switch component is a **dropdown menu** that **contains all the languages** of the website.

![Mobile language switch component](../../images/leptonxlite-mobile-language-switch-component.png)

How to override the Mobile Language Switch Component in LeptonX Lite MVC

* The **mobile language switch component page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/MobileLanguageSwitch/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **mobile language switch component (C#)** is defined in the `Themes/LeptonXLite/Components/MobileLanguageSwitch/MobileLanguageSwitchViewComponent.cs` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

User Menu Component

The **User Menu** is the **menu** that **drops down** when you **click your name** or **profile picture** in the **upper right corner** of your page (**in the toolbar**). It drops down options such as **Settings**, **Logout**, etc.

![User menu component](../../images/leptonxlite-user-menu-component.png)

How to override the User Menu Component in LeptonX Lite MVC

* The **user menu component page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/UserMenu/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **user menu component (C#)** is defined in the `Themes/LeptonXLite/Components/UserMenu/UserMenuViewComponent.cs` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

Mobile User Menu Component

The **mobile user menu component** is used to display the **user menu on mobile devices**. The mobile user menu component is a **dropdown menu** that contains all the **options** of the **user menu**.

![Mobile user menu component](../../../../images/leptonxlite-mobile-user-menu-component.png)

How to override the Mobile User Menu Component in LeptonX Lite MVC

* The **mobile user menu page (.cshtml file)** is defined in the `Themes/LeptonXLite/Components/MobileUserMenu/Default.cshtml` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

* The **mobile user menu component (C#)** is defined in the `Themes/LeptonLite/Components/MobileUserMenu/MobileUserMenuViewComponent.cs` file and you can **override it** by creating a file with the **same name** and **under** the **same folder**.

10.1.20 JavaScript API

10.1.20.1 Overall

JavaScript API

ABP provides a set of JavaScript APIs for ASP.NET Core MVC / Razor Pages applications. They can be used to perform common application requirements easily in the client side and integrate to the server side.

APIs

- * [AJAX] (Ajax.md)
- * [Auth] (Auth.md)
- * [CurrentUser] (CurrentUser.md)
- * [DOM] (DOM.md)
- * [Events] (Events.md)
- * [Features] (Features.md)
- * [Global Features] (GlobalFeatures.md)
- * [Localization] (Localization.md)
- * [Logging] (Logging.md)
- * [ResourceLoader] (ResourceLoader.md)
- * [Settings] (Settings.md)
- * [UI Block/Busy] (Block-Busy.md)
- * [UI Message] (Message.md)
- * [UI Notification] (Notify.md)

10.1.20.2 Localization

ASP.NET Core MVC / Razor Pages UI: JavaScript Localization API

Localization API allows you to reuse the server side localization resources in the client side.

> This document only explains the JavaScript API. See the [localization document](../../../../Localization.md) to understand the ABP localization system.

Basic Usage

``abp.localization.getResource(...)`` function is used to get a localization resource:

```
```js
var testResource = abp.localization.getResource('Test');
````
```

Then you can localize a string based on this resource:

```
```js
var str = testResource('HelloWorld');
````
```

``abp.localization.localize(...)`` function is a shortcut where you can both specify the text name and the resource name:

```
```js
var str = abp.localization.localize('HelloWorld', 'Test');
````
```

``HelloWorld`` is the text to localize, where ``Test`` is the localization resource name here.

Fallback Logic

If given texts was not localized, localization method returns the given key as the localization result.

Default Localization Resource

If you don't specify the localization resource name, it uses the ****default localization resource**** defined on the ``AbpLocalizationOptions`` (see the [localization document](../../../../Localization.md)).

****Example: Using the default localization resource****

```
```js
var str = abp.localization.localize('HelloWorld'); //uses the default resource
````
```

Format Arguments

If your localized string contains arguments, like ``Hello {0}, welcome!``, you can pass arguments to the localization methods. Examples:

```
```js
var testSource = abp.localization.getResource('Test');
var str1 = testSource('HelloWelcomeMessage', 'John');
````
```

```
var str2 = abp.localization.localize('HelloWelcomeMessage', 'Test', 'John');
````
```

Assuming the `HelloWelcomeMessage` is localized as `Hello {0}, welcome!`, both of the samples above produce the output `Hello John, welcome!`.

## ## Other Properties & Methods

### ### abp.localization.resources

`abp.localization.resources` property stores all the localization resources, keys and their values.

### ### abp.localization.isLocalized

Returns a boolean indicating that if the given text was localized or not.

#### \*\*Example\*\*

```
```js
abp.localization.isLocalized('ProductName', 'MyResource');
````
```

Returns `true` if the `ProductName` text was localized for the `MyResource` resource. Otherwise, returns `false`. You can leave the resource name empty to use the default localization resource.

### ### abp.localization.defaultResourceName

`abp.localization.defaultResourceName` can be set to change the default localization resource. You normally don't set this since the ABP Framework automatically sets it based on the server side configuration.

### ### abp.localization.currentCulture

`abp.localization.currentCulture` returns an object to get information about the **currently selected language**.

An example value of this object is shown below:

```
```js
{
  "displayName": "English",
  "englishName": "English",
  "threeLetterIsoLanguageName": "eng",
  "twoLetterIsoLanguageName": "en",
  "isRightToLeft": false,
  "cultureName": "en",
  "name": "en",
  "nativeName": "English",
  "dateTimeFormat": {
    "calendarAlgorithmType": "SolarCalendar",
    "dateSeparator": ".",
    "timeSeparator": ":",
    "longDatePattern": "yyyy-MM-dd'T'HH:mm:ss",
    "shortDatePattern": "yyyy-MM-dd"
  }
}
````
```

```

 "dateTimeFormatLong": "dddd, MMMM d, yyyy",
 "shortDatePattern": "M/d/yyyy",
 "fullDateTimePattern": "dddd, MMMM d, yyyy h:mm:ss tt",
 "dateSeparator": "/",
 "shortTimePattern": "h:mm tt",
 "longTimePattern": "h:mm:ss tt"
}
}
```

```

abp.localization.languages

Used to get list of all ****available languages**** in the application. An example value of this object is shown below:

```

```
js
[
{
 "cultureName": "en",
 "uiCultureName": "en",
 "displayName": "English",
 "flagIcon": null
},
{
 "cultureName": "fr",
 "uiCultureName": "fr",
 "displayName": "Français",
 "flagIcon": null
},
{
 "cultureName": "pt-BR",
 "uiCultureName": "pt-BR",
 "displayName": "Português",
 "flagIcon": null
},
{
 "cultureName": "tr",
 "uiCultureName": "tr",
 "displayName": "Türkçe",
 "flagIcon": null
},
{
 "cultureName": "zh-Hans",
 "uiCultureName": "zh-Hans",
 "displayName": "简体中文",
 "flagIcon": null
}
]
```

```

10.1.20.3 Auth

ASP.NET Core MVC / Razor Pages UI: JavaScript Auth API

Auth API allows you to check permissions (policies) for the current user in the client side. In this way, you can conditionally show/hide UI parts or perform your client side logic based on the current permissions.

> This document only explains the JavaScript API. See the [authorization document](../../../../Authorization.md) to understand the ABP authorization & permission system.

Basic Usage

```abp.auth.isGranted(...)`` function is used to check if a permission/policy has granted or not:

```
```js
if (abp.auth.isGranted('DeleteUsers')) {
    //TODO: Delete the user
} else {
    alert("You don't have permission to delete a user!");
}
````
```

#### ## Other Fields & Functions

- \* ```abp.auth.isAnyGranted(...)`` : Gets one or more permission/policy names and returns `true` if at least one of them has granted.
- \* ```abp.auth.areAllGranted(...)`` : Gets one or more permission/policy names and returns `true` if all of them of them have granted.
- \* ```abp.auth.grantedPolicies`` : This is an object where its keys are the permission/policy names. You can find the granted permission/policy names here.

### 10.1.20.4 Current User

#### # ASP.NET Core MVC / Razor Pages UI: JavaScript CurrentUser API

```abp.currentUser`` is an object that contains information about the current user of the application.

> This document only explains the JavaScript API. See the [CurrentUser document](../../../../CurrentUser.md) to get information about the current user in the server side.

Authenticated User

If the user was authenticated, this object will be something like below:

```
```js
```

```
{
 isAuthenticated: true,
 id: "34f1f4a7-13cc-4b91-84d1-b91c87afa95f",
 tenantId: null,
 userName: "john",
 name: "John",
 surName: "Nash",
 email: "john.nash@abp.io",
 emailVerified: true,
 phoneNumber: null,
 phoneNumberVerified: false,
 roles: ["moderator", "supporter"]
}
````
```

So, `abp.currentUser.userName` returns `john` in this case.

Anonymous User

If the user was not authenticated, this object will be something like below:

```
```js
{
 isAuthenticated: false,
 id: null,
 tenantId: null,
 userName: null,
 name: null,
 surName: null,
 email: null,
 emailVerified: false,
 phoneNumber: null,
 phoneNumberVerified: false,
 roles: []
}
````
```

You can check `abp.currentUser.isAuthenticated` to understand if the user was authenticated or not.

10.1.20.5 Settings

ASP.NET Core MVC / Razor Pages UI: JavaScript Setting API

Localization API allows you to get the values of the settings on the client side. You can read the current value of a setting in the client side only if it is allowed by the setting definition (on the server side).

> This document only explains the JavaScript API. See the [settings document](../../../../Settings.md) to understand the ABP setting system.

```

## Basic Usage

```js
//Gets a value as string.
var language = abp.setting.get('Abp.Localization.DefaultLanguage');

//Gets an integer value.
var requiredLength = abp.setting.getInt('Abp.Identity.Password.RequiredLength');

//Gets a boolean value.
var requireDigit = abp.setting.getBoolean('Abp.Identity.Password.RequireDigit');
```

```

All Values

``abp.setting.values`` can be used to obtain all the setting values as an object where the object properties are setting names and property values are the setting values.

An example value of this object is shown below:

```

```js
{
 Abp.Localization.DefaultLanguage: "en",
 Abp.Timing.TimeZone: "UTC",
 ...
}
```

```

10.1.20.6 Features

ASP.NET Core MVC / Razor Pages UI: JavaScript Features API

``abp.features`` API allows you to check features or get the values of the features on the client side. You can read the current value of a feature in the client side only if it is allowed by the feature definition (on the server side).

> This document only explains the JavaScript API. See the [Features](../../../../Features.md) document to understand the ABP Features system.

Basic Usage

```

```js
//Gets a value as string.
var value = abp.features.get('ExportingToExcel');

//Check the feature is enabled
var enabled = abp.features.isEnabled('ExportingToExcel.Enabled');
```

```

All Values

``abp.features.values`` can be used to access to the all feature values.

An example value of this object is shown below:

```
```js
{
 Identity.TwoFactor: "Optional",
 ExportingToExcel.Enabled: "true",
 ...
}
````
```

10.1.20.7 AJAX

ASP.NET Core MVC / Razor Pages UI JavaScript AJAX API

``abp.ajax`` API provides a convenient way of performing AJAX calls to the server. It internally uses JQuery's ``$.ajax``, but automates some common tasks for you;

- * Automatically **handles & localize the errors** and informs the user (using the [`abp.message`](Message.md)). So you typically don't care about errors.
- * Automatically adds **anti forgery** token to the HTTP header to satisfy CSRF protection validation on the server side.
- * Automatically sets **default options** and allows to configure the defaults in a single place.
- * Can **block** a UI part (or the full page) during the AJAX operation.
- * Allows to fully customize any AJAX call, by using the standard ``$.ajax` **options**`.`

> While ``abp.ajax`` makes the AJAX call pretty easier, you typically will use the [`Dynamic JavaScript Client Proxy`](../Dynamic-JavaScript-Proxies.md) system to perform calls to your server side HTTP APIs. ``abp.ajax`` can be used when you need to perform low level AJAX operations.

Basic Usage

``abp.ajax`` accepts an options object that is accepted by the standard `[\$.ajax](https://api.jquery.com/jquery.ajax/#jQuery-ajax-settings)`. All the standard options are valid. It returns a [`promise`](https://api.jquery.com/category/deferred-object/) as the return value.

Example: Get the list of users

```
```js
abp.ajax({
 type: 'GET',
 url: '/api/identity/users'
}).then(function(result) {
 console.log(result);
});
````
```

~~~~

This command logs the list of users to the console, if you've **logged in** to the application and have [permission](../../../../Authorization.md) for the user management page of the [Identity Module](../../../../Modules/Identity.md).

## ## Error Handling

The example AJAX call above shows an **error message** if you haven't login to the application or you don't have the necessary permissions to perform this request:

```
![ajax-error](../../../../images/ajax-error.png)
```

All kinds of errors are automatically handled by `abp.ajax`, unless you want to disable it.

### ### Standard Error Response

`abp.ajax` is compatible with the [exception handling system](../../../../Exception-Handling.md) of the ABP Framework and it properly handles the standard error format returned from the server. A typical error message is a JSON as like below:

```
~~~~ json
{
 "error": {
 "code": "App:010042",
 "message": "This topic is locked and can not add a new message",
 "details": "A more detailed info about the error..."
 }
}
~~~~
```

The error message is directly shown to the user, using the `message` and `details` properties.

### ### Non-Standard Error Response & HTTP Status Codes

It also handles errors even if the standard error format was not sent by the server. This can be case if you bypass the ABP exception handling system and manually build the HTTP response on the server. In that case, **HTTP status codes** are considered.

The following HTTP Status Codes are pre-defined;

- \* **401**: Shows an error message like "*You should be authenticated (sign in) in order to perform this operation*". When the users click the OK button, they are redirected to the home page of the application to make them login again.
- \* **403**: Shows an error message like "*You are not allowed to perform this operation*".
- \* **404**: Shows an error message like "*The resource requested could not found on the server*".
- \* **Others**: Shows a generic error message like "*An error has occurred. Error detail not sent by server*".

All these messages are localized based on the current user's language.

#### #### Manually Handling the Errors

Since `abp.ajax` returns a promise, you can always chain a ` `.catch(...)` call to register a callback that is executed if the AJAX request fails.

##### **\*\*Example: Show an alert if the AJAX request fails\*\***

```
```js
abp.ajax({
    type: 'GET',
    url: '/api/identity/users'
}).then(function(result) {
    console.log(result);
}).catch(function() {
    alert("request failed :(");
});
````
```

While your callback is fired, ABP still handles the error itself. If you want to disable automatic error handling, pass `abpHandleError: false` the the `abp.ajax` options.

##### **\*\*Example: Disable the auto error handling\*\***

```
```js
abp.ajax({
    type: 'GET',
    url: '/api/identity/users',
    abpHandleError: false //DISABLE AUTO ERROR HANDLING
}).then(function(result) {
    console.log(result);
}).catch(function() {
    alert("request failed :(");
});
````
```

If you set `abpHandleError: false` and don't catch the error yourself, then the error will be hidden and the request silently fails. `abp.ajax` still logs the error to the browser console (see the *\*Configuration\** section to override it).

#### ## Configuration

`abp.ajax` has a **\*\*global configuration\*\*** that you can customize based on your requirements.

#### #### Default AJAX Options

`abp.ajax.defaultOpts` object is used to configure default options used while performing an AJAX call, unless you override them. Default value of this object is shown below:

```
```js
```

```
{
  dataType: 'json',
  type: 'POST',
  contentType: 'application/json',
  headers: {
    'X-Requested-With': 'XMLHttpRequest'
  }
}
````
```

So, if you want to change the default request type, you can do it as shown below:

```
````js
abp.ajax.defaultOpts.type = 'GET';
````
```

Write this code before all of your JavaScript code. You typically want to place such a configuration into a separate JavaScript file and add it to the layout using the global [bundle](./Bundling-Minification.md).

#### ### Log/Show Errors

The following functions can be overridden to customize the logging and showing the error messages:

- \* ``abp.ajax.logError` function logs errors using the [abp.log.error(...)](Logging.md) by default.
- \* ``abp.ajax.showError` function shows the error message using the [abp.message.error(...)](Message.md) by default.
- \* ``abp.ajax.handleErrorStatusCode` handles different HTTP status codes and shows different messages based on the code.
- \* ``abp.ajax.handleAbpErrorResponse` handles the errors sent with the standard ABP error format.
- \* ``abp.ajax.handleNonAbpErrorResponse` handles the non-standard error responses.
- \* ``abp.ajax.handleUnauthorizedRequest` handles responses with `401` status code and redirect users to the home page of the application.

#### **\*\*Example: Override the `logError` function\*\***

```
````js
abp.ajax.logError = function(error) {
  //...
}
````
```

#### ### Other Options

- \* ``abp.ajax.ajaxSendHandler` function is used to intercept the AJAX requests and add antiforgery token to the HTTP header. Note that this works for all AJAX requests, even if you don't use the ``abp.ajax`.

#### 10.1.20.8 Message

```
# ASP.NET Core MVC / Razor Pages UI: JavaScript Message API
```

Message API is used to show nice looking messages to the user as a blocking dialog. Message API is an abstraction provided by the ABP Framework and implemented using the [[SweetAlert](#)] (<https://sweetalert.js.org/>) library by default.

**## Quick Example**

Use `abp.message.success(...)` function to show a success message:

```
```js
abp.message.success('Your changes have been successfully saved!', 'Congratulations');
````
```

It will show a dialog on the UI:

```
![js=success](../../../../images/js-success.png)
```

**## Informative Messages**

There are four types of informative message functions:

- \* `abp.message.info(...)`
- \* `abp.message.success(...)`
- \* `abp.message.warn(...)`
- \* `abp.message.error(...)`

All these methods get two parameters:

- \* `message`: The message (`string`) to be shown.
- \* `title`: An optional (`string`) title.

**\*\*Example: Show an error message\*\***

```
```js
abp.message.error('Your credit card number is not valid!');
````
```

```
![js=error](../../../../images/js-error.png)
```

**## Confirmation Message**

`abp.message.confirm(...)` function can be used to get a confirmation from the user.

**\*\*Example\*\***

Use the following code to get a confirmation result from the user:

```
```js
```

```

abp.message.confirm('Are you sure to delete the "admin" role?')
.then(function(confirmed) {
  if(confirmed) {
    console.log('TODO: deleting the role...');
  }
});
```

```

The resulting UI will be like shown below:

![[js=confirm]](../../../../../images/js-message-confirm.png)

If user has clicked the `Yes` button, the `confirmed` argument in the `then` callback function will be `true`.

> *"Are you sure?"* is the default title (localized based on the current language) and you can override it.

#### ### The Return Value

The return value of the `abp.message.confirm(...)` function is a promise, so you can chain a `then` callback as shown above.

#### ### Parameters

`abp.message.confirm(...)` function has the following parameters:

- \* `message`: A message (string) to show to the user.
- \* `titleOrCallback` (optional): A title or a callback function. If you supply a string, it is shown as the title. If you supply a callback function (that gets a `bool` parameter) then it's called with the result.
- \* `callback` (optional): If you've passes a title to the second parameter, you can pass your callback function as the 3rd parameter.

Passing a callback function is an alternative to the `then` callback shown above.

#### **\*\*Example: Providing all the parameters and getting result with the callback function\*\***

```

```js
abp.message.confirm(
  'Are you sure to delete the "admin" role?',
  'Be careful!',
  function(confirmed) {
    if(confirmed) {
      console.log('TODO: deleting the role...');

    }
  });
```

```

#### ## SweetAlert Configuration

The Message API is implemented using the [SweetAlert] (<https://sweetalert.js.org/>) library by default. If you want to change its configuration, you can set the options in the `abp.libs/sweetalert.config` object. The default configuration object is shown below:

```
```js
{
  'default': {
  },
  info: {
    icon: 'info'
  },
  success: {
    icon: 'success'
  },
  warn: {
    icon: 'warning'
  },
  error: {
    icon: 'error'
  },
  confirm: {
    icon: 'warning',
    title: 'Are you sure?',
    buttons: ['Cancel', 'Yes']
  }
}
````
```

> "Are you sure?", "Cancel" and "Yes" texts are automatically localized based on the current language.

So, if you want to set the `warn` icon, you can set it like:

```
```js
abp.libs/sweetalert.config.warn.icon = 'error';
````
```

See the [SweetAlert document] (<https://sweetalert.js.org/>) for all the configuration options.

#### 10.1.20.9 Notify

```
# ASP.NET Core MVC / Razor Pages UI: JavaScript Notify API
```

Notify API is used to show toast style, auto disappearing UI notifications to the end user. It is implemented by the [ Toastr] (<https://github.com/CodeSeven/toastr>) library by default.

```
## Quick Example
```

Use `abp.notify.success(...)` function to show a success message:

```
```js
abp.notify.success(
    'The product "Acme Atom Re-Arranger" has been successfully deleted.',
    'Deleted the Product'
);
````
```

A notification message is shown at the bottom right of the page:

```
![js=success]../../images/js-notify-success.png)
```

## ## Notification Types

There are four types of pre-defined notifications;

- \* `abp.notify.success(...)`
- \* `abp.notify.info(...)`
- \* `abp.notify.warn(...)`
- \* `abp.notify.error(...)`

All of the methods above gets the following parameters;

- \* `message`: A message (`string`) to show to the user.
- \* `title`: An optional title (`string`).
- \* `options`: Additional options to be passed to the underlying library, to the Toastr by default.

## ## Toastr Configuration

The notification API is implemented by the [Toastr](<https://github.com/CodeSeven/toastr>) library by default. You can see its own configuration options.

### \*\*Example: Show toast messages on the top right of the page\*\*

```
```js
toastr.options.positionClass = 'toast-top-right';
````
```

> ABP sets this option to `toast-bottom-right` by default. You can override it just as shown above.

## 10.1.20.10 Block/Busy

### # ASP.NET Core MVC / Razor Pages UI: JavaScript UI Block/Busy API

UI Block API disables (blocks) the page or a part of the page.

## ## Basic Usage

**\*\*Example: Block (disable) the complete page\*\***

```
```js
abp.ui.block();
````
```

**\*\*Example: Block (disable) an HTML element\*\***

```
```js
abp.ui.block('#MyContainer');
````
```

**\*\*Example: Enables the previously blocked element or page:\*\***

```
```js
abp.ui.unblock();
````
```

**## Options**

``abp.ui.block()`` method can get an options object which may contain the following fields:

- \* `elm`: An optional selector to find the element to be blocked (e.g. `'#MyContainerId}`). If not provided, the entire page is blocked. The selector can also be directly passed to the `block()` method as shown above.
- \* `busy`: Set to `true` to show a progress indicator on the blocked area.
- \* `promise`: A promise object with `always` or `finally` callbacks. This can be helpful if you want to automatically unblock the blocked area when a deferred operation completes.

**\*\*Example: Block an element with busy indicator\*\***

```
```js
abp.ui.block({
  elm: '#MySection',
  busy: true
});
````
```

The resulting UI will look like below:

```
![ui=busy](../../../../images/ui-busy.png)
```

**## setBusy**

``abp.ui.setBusy(...)`` and ``abp.ui.clearBusy()`` are shortcut functions if you want to use the block with `busy` option.

**\*\*Example: Block with busy\*\***

```
```js
abp.ui.setBusy('#MySection');
````
```

```
```
```

Then you can use `abp.ui.clearBusy()` to re-enable the busy area/page.

### 10.1.20.11 Events

#### # ASP.NET Core MVC / Razor Pages UI: JavaScript Events API

`abp.event` object is a simple service that is used to publish and subscribe to global events **\*\*in the browser\*\***.

➤ This API is not related to server side local or distributed events. It works in the browser boundaries to make the UI components (code parts) communicate in a loosely coupled way.

##### ## Basic Usage

###### ### Publishing Events

Use `abp.event.trigger` to publish events.

**\*\*Example: Publish a *Basket Updated* event\*\***

```
```js
```

```
abp.event.trigger('basketUpdated');
```

```
```
```

This will trigger all the subscribed callbacks.

###### ### Subscribing to the Events

Use `abp.event.on` to subscribe to events.

**\*\*Example: Consume the *Basket Updated* event\*\***

```
```js
```

```
abp.event.on('basketUpdated', function() {
  console.log('Handled the basketUpdated event...');
});
```

```
```
```

You start to get events after you subscribe to the event.

###### ### Unsubscribing from the Events

If you need to unsubscribe from a pre-subscribed event, you can use the `abp.event.off(eventName, callback)` function. In this case, you have the callback as a separate function declaration.

**\*\*Example: Subscribe & Unsubscribe\*\***

```

```js
function onBasketUpdated() {
    console.log('Handled the basketUpdated event...');
}

//Subscribe
abp.event.on('basketUpdated', onBasketUpdated);

//Unsubscribe
abp.event.off('basketUpdated', onBasketUpdated);
```

```

You don't get events after you unsubscribe from the event.

#### ## Event Arguments

You can pass arguments (of any count) to the `trigger` method and get them in the subscription callback.

#### **\*\*Example: Add the basket as the event argument\*\***

```

```js
//Subscribe to the event
abp.event.on('basketUpdated', function(basket) {
    console.log('The new basket object: ');
    console.log(basket);
});

//Trigger the event
abp.event.trigger('basketUpdated', {
    items: [
        {
            "productId": "123",
            "count": 2
        },
        {
            "productId": "832",
            "count": 1
        }
    ]
});
```

```

#### ### Multiple Arguments

If you want to pass multiple arguments, you can pass like `abp.event.on('basketUpdated', arg0, arg1, agr2)`. Then you can add the same argument list to the callback function on the subscriber side.

> **\*\*Tip:\*\*** Alternatively, you can send a single object that has a separate field for each argument. This makes easier to extend/change the event arguments in the future without breaking the subscribers.

## 10.1.20.12 DOM

```
# ASP.NET Core MVC / Razor Pages UI: JavaScript DOM API
```

``abp.dom`` (Document Object Model) provides events that you can subscribe to get notified when elements dynamically added to and removed from the page (DOM).

It is especially helpful if you want to initialize the new loaded elements. This is generally needed when you dynamically add elements to DOM (for example, get some HTML elements via AJAX) after page initialization.

> ABP uses the [MutationObserver] (<https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver>) to observe the changes made on the DOM.

```
## Node Events
```

```
#### onNodeAdded
```

This event is triggered when an element is added to the DOM. Example:

```
```js
abp.dom.onNodeAdded(function(args) {
    console.log(args.$el);
});
````
```

``args`` object has the following fields;

\* ` `\$el` `: The JQuery selection to get the new element inserted to the DOM.

```
#### onNodeRemoved
```

This event is triggered when an element is removed from the DOM. Example:

```
```js
abp.dom.onNodeRemoved(function(args) {
    console.log(args.$el);
});
````
```

``args`` object has the following fields;

\* ` `\$el` `: The JQuery selection to get the element removed from the DOM.

```
## Pre-Build Initializers
```

ABP Framework is using the DOM events to initialize some kind of HTML elements when they are added to the DOM after than the page was already initialized.

› Note that the same initializers also work if these elements were already included in the initial DOM. So, whether they are initially or lazy loaded, they work as expected.

### ### Form Initializer

The Form initializer (defined as `abp.dom.initializers.initializeForms`) initializes the lazy loaded forms;

- \* Automatically enabled the `unobtrusive` validation on the form.
- \* Can automatically show a confirmation message when you submit the form. To enable this feature, just add `data-confirm` attribute with a message (like `data-confirm="Are you sure?") to the `form` element.
- \* If the `form` element has `data-ajaxForm="true"` attribute, then automatically calls the `\$.abpAjaxForm()` on the `form` element, to make the form posted via AJAX.

See the [Forms & Validation](./Forms-Validation.md) document for more.

### ### Script Initializer

Script initializer (`abp.dom.initializers.initializeScript`) can execute a JavaScript code for a DOM element.

**\*\*Example: Lazy load a component and execute some code when the element has loaded\*\***

Assume that you've a container to load the element inside:

```
```html
<div id="LazyComponent"></div>
```
```

And this is the component that will be loaded via AJAX from the server and inserted into the container:

```
```html
<div data-script-class="MyCustomClass">
    <p>Sample message</p>
</div>
```
```

`data-script-class="MyCustomClass"` indicates the JavaScript class that will be used to perform some logic on this element:

`MyCustomClass` is a global object defined as shown below:

```
```js
MyCustomClass = function() {

    function initDom($el) {
        $el.css('color', 'red');
    }
}
```

```
        return {
            initDom: initDom
        }
    };
}
```

`initDom` is the function that is called by the ABP Framework. The ` `\$el` argument is the loaded HTML element as a JQuery selection.

Finally, you can load the component inside the container after an AJAX call:

```
```js
$(function () {
    setTimeout(function () {
        $.get('/get-my-element').then(function (response) {
            $('#LazyComponent').html(response);
        });
    }, 2000);
});
```

```

Script Initialization system is especially helpful if you don't know how and when the component will be loaded into the DOM. This can be possible if you've developed a reusable UI component in a library and you want the application developer shouldn't care how to initialize the component in different use cases.

> Script initialization doesn't work if the component was loaded in the initial DOM. In this case, you are responsible to initialize it.

#### ### Other Initializers

The following Bootstrap components and libraries are automatically initialized when they are added to the DOM:

- \* Tooltip
- \* Popover
- \* Timeago

### 10.1.20.13 Logging

#### # ASP.NET Core MVC / Razor Pages UI: JavaScript Logging API

`abp.log` API is used to write simple logs in the client side.

> The logs are written to console, using the `console.log`, by default.

> This document is for simple client side logging. See the [Logging](../../../../Logging.md) document for server side logging system.

#### ## Basic Usage

Use one of the ``abp.log.xxx(...)`` methods based on the severity of your log message.

```
```js
abp.log.debug("Some debug log here..."); //Logging a simple debug message
abp.log.info({ name: "john", age: 42 }); //Logging an object as an information log
abp.log.warn("A warning message"); //Logging a warning message
abp.log.error('An error happens...'); //Error message
abp.log.fatal('Network connection has gone away!'); //Fatal error
```
```

## ## Log Levels

There are 5 levels for a log message:

- \* DEBUG = 1
- \* INFO = 2
- \* WARN = 3
- \* ERROR = 4
- \* FATAL = 5

These are defined in the ``abp.log.levels`` object (like ``abp.log.levels.WARN``).

## ### Changing the Current Log Level

You can control the log level as shown below:

```
```js
abp.log.level = abp.log.levels.WARN;
```
```

Default log level is ``DEBUG``.

## ### Logging with Specifying the Level

Instead of calling ``abp.log.info(...)`` function, you can use the ``abp.log.log`` by specifying the log level as a parameter:

```
```js
abp.log.log("log message...", abp.log.levels.INFO);
```
```

## 10.1.20.14      *Resource Loader*

### # ASP.NET Core MVC / Razor Pages UI: JavaScript Resource Loader API

``abp.ResourceLoader`` is a service that can load a JavaScript or CSS file on demand. It guarantees to load the file only once even if you request multiple times.

## ## Loading Script Files

``abp.ResourceLoader.loadScript(...)`` function **\*\*loads\*\*** a JavaScript file from the server and **\*\*executes\*\*** it.

### **\*\*Example: Load a JavaScript file\*\***

```
```js
abp.ResourceLoader.loadScript('/Pages/my-script.js');
```
```

### ### Parameters

``loadScript`` function can get three parameters;

- \* `url` (required, `string`): The URL of the script file to be loaded.
- \* `loadCallback` (optional, `function`): A callback function that is called once the script is loaded & executed. In this callback you can safely use the code in the script file. This callback is called even if the file was loaded before.
- \* `failCallback` (optional, `function`): A callback function that is called if loading the script fails.

### **\*\*Example: Provide the `loadCallback` argument\*\***

```
```js
abp.ResourceLoader.loadScript('/Pages/my-script.js', function() {
    console.log('successfully loaded :)');
});
```
```

## ## Loading Style Files

``abp.ResourceLoader.loadStyle(...)`` function adds a `link` element to the `head` of the document for the given URL, so the CSS file is automatically loaded by the browser.

### **\*\*Example: Load a CSS file\*\***

```
```js
abp.ResourceLoader.loadStyle('/Pages/my-styles.css');
```
```

## 10.1.21 Customize/Extend the UI

### *10.1.21.1 Overall*

#### # ASP.NET Core (MVC / Razor Pages) User Interface Customization Guide

This document explains how to override the user interface of a depended [application module](../../Modules/Index.md) or [theme](Theming.md) for ASP.NET Core MVC / Razor Page applications.

## ## Overriding a Page

This section covers the [Razor Pages] (<https://docs.microsoft.com/en-us/aspnet/core/razor-pages/>) development, which is the recommended approach to create server rendered user interface for ASP.NET Core. Pre-built modules typically uses the Razor Pages approach instead of the classic MVC pattern (next sections will cover the MVC pattern too).

You typically have three kind of override requirement for a page:

- \* Overriding **\*\*only the Page Model\*\*** (C#) side to perform additional logic without changing the page UI.
- \* Overriding **\*\*only the Razor Page\*\*** (.chtml file) to change the UI without changing the c# behind the page.
- \* **\*\*Completely overriding\*\*** the page.

### #### Overriding a Page Model (C#)

```
```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Volo.Abp.DependencyInjection;
using Volo.Abp.Identity;
using Volo.Abp.Identity.Web.Pages.Identity.Users;

namespace Acme.BookStore.Web.Pages.Identity.Users
{
    [Dependency(ReplaceServices = true)]
    [ExposeServices(typeof(EditModalModel))]
    public class MyEditModalModel : EditModalModel
    {
        public MyEditModalModel(
            IIdentityUserAppService identityUserAppService,
            IIdentityRoleAppService identityRoleAppService
        ) : base(
            identityUserAppService,
            identityRoleAppService)
        {
        }

        public async override Task<IActionResult> OnPostAsync()
        {
            //TODO: Additional logic
            await base.OnPostAsync();
            //TODO: Additional logic
        }
    }
}
```

\* This class inherits from and replaces the `EditModalModel` for the users and overrides the `OnPostAsync` method to perform additional logic before and after the underlying code.

\* It uses `ExposeServices` and `Dependency` attributes to replace the class.

#### ### Overriding a Razor Page (.CSHTML)

Overriding a `\*.cshtml` file (razor page, razor view, view component... etc.) is possible through creating the same `\*.cshtml` file under the same path.

#### #### Example

This example overrides the **\*\*login page\*\*** UI defined by the [Account Module](../../Modules/Account.md).

The account module defines a `Login.cshtml` file under the `Pages/Account` folder. So, you can override it by creating a file in the same path:

```
![overriding-login-cshtml](../../images/overriding-login-cshtml.png)
```

You typically want to copy the original `\*.cshtml` file of the module, then make the necessary changes. You can find the original file [here](https://github.com/abpframework/abp/blob/dev/modules/account/src/Volo.Abp.Account.Web/Pages/Account/Login.cshtml). Do not copy the `Login.cshtml.cs` file which is the code behind file for the razor page and we don't want to override it yet (see the next section).

> Don't forget to add [ViewImports.cshtml](https://learn.microsoft.com/en-us/aspnet/core/mvc/views/layout?view=aspnetcore-7.0#importing-shared-directives) if the page you want to override contains [ABP Tag Helpers](../Tag-Helpers/Index.md).

```
```csharp
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, Volo.Abp.AspNetCore.Mvc.UI
@addTagHelper *, Volo.Abp.AspNetCore.Mvc.UI.Bootstrap
@addTagHelper *, Volo.Abp.AspNetCore.Mvc.UI.Bundling
```
```

That's all, you can change the file content however you like.

#### ### Completely Overriding a Razor Page

You may want to completely override a page; the razor and the c# file related to the page.

In such a case;

1. Override the C# page model class just like described above, but don't replace the existing page model class.
2. Override the Razor Page just described above, but also change the @model directive to point your new page model.

#### #### Example

This example overrides the **\*\*login page\*\*** defined by the [Account Module](../../Modules/Account.md).

Create a page model class deriving from the `LoginModel` (defined in the `Volo.Abp.Account.Web.Pages.Account` namespace):

```
```csharp
public class MyLoginModel : LoginModel
{
    public MyLoginModel(
        IAuthenticationSchemeProvider schemeProvider,
        IOptions<AbpAccountOptions> accountOptions
    ) : base(
        schemeProvider,
        accountOptions)
    {

    }

    public override Task<IActionResult> OnPostAsync(string action)
    {
        //TODO: Add logic
        return base.OnPostAsync(action);
    }

    //TODO: Add new methods and properties...
}
````
```

You can override any method or add new properties/methods if needed.

> Notice that we didn't use `[Dependency(ReplaceServices = true)]` or `[ExposeServices(typeof(LoginModel))]` since we don't want to replace the existing class in the dependency injection, we define a new one.

Copy `Login.cshtml` file into your solution as just described above. Change the \*\*@model\*\* directive to point to the `MyLoginModel`:

```
```xml
@page
...
@model Acme.BookStore.Web.Pages.Account.MyLoginModel
...````
```

That's all! Make any change in the view and run your application.

#### #### Replacing Page Model Without Inheritance

You don't have to inherit from the original page model class (like done in the previous example). Instead, you can completely \*\*re-implement\*\* the page yourself. In this case, just derive from `PageModel`, `AbpPageModel` or any suitable base class you need.

#### ## Overriding a View Component

The ABP Framework, pre-built themes and modules define some **\*\*re-usable view components\*\***. These view components can be replaced just like a page described above.

### ### Example

The screenshot below was taken from the [Basic Theme](Basic-Theme.md) comes with the application startup template.

![bookstore-brand-area-highlighted](../../images/bookstore-brand-area-highlighted.png)

The [Basic Theme](Basic-Theme.md) defines some view components for the layout. For example, the highlighted area with the red rectangle above is called **\*\*Brand component\*\***. You probably want to customize this component by adding your **\*\*own application logo\*\***. Let's see how to do it.

First, create your logo and place under a folder in your web application. We used `wwwroot/logos/bookstore-logo.png` path. Then copy the Brand component's view ([from here](https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic/Themes/Basic/Components/Brand/Default.cshtml)) from the basic theme files under the `Themes/Basic/Components/Brand` folder. The result should be similar the picture below:

![bookstore-added-brand-files](../../images/bookstore-added-brand-files.png)

Then change the `Default.cshtml` as you like. Example content can be like that:

```
```xml
<a href="/">
    
</a>
```
```

Now, you can run the application to see the result:

![bookstore-added-logo](../../images/bookstore-added-logo.png)

If you need, you can also replace [the code behind c# class](https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic/Themes/Basic/Components/Brand/MainNavbarBrandViewComponent.cs) of the component just using the dependency injection system.

### ### Overriding the Theme

Just as explained above, you can replace any component, layout or c# class of the used theme. See the [theming document](Theming.md) for more information on the theming system.

## ## Overriding Static Resources

Overriding a static embedded resource (like JavaScript, Css or image files) of a module is pretty easy. Just place a file in the same path in your solution and let the [Virtual File System](../../Virtual-File-System.md) to handle it.

## ## Manipulating the Bundles

The [Bundling & Minification](Bundling-Minification.md) system provides an **extensible and dynamic** system to create **script** and **style** bundles. It allows you to extend and manipulate the existing bundles.

### #### Example: Add a Global CSS File

For example, ABP Framework defines a **global style bundle** which is added to every page (actually, added to the layout by the themes). Let's add a **custom style file** to the end of the bundle files, so we can override any global style.

First, create a CSS file and locate it in a folder inside the `wwwroot`:

```
![bookstore-global-css-file](../../images/bookstore-global-css-file.png)
```

Define some custom CSS rules inside the file. Example:

```
```css
.card-title {
    color: orange;
    font-size: 2em;
    text-decoration: underline;
}

.btn-primary {
    background-color: red;
}
````
```

Then add this file to the standard global style bundle in the `ConfigureServices` method of your [module](../../Module-Development-Basics.md):

```
```csharp
Configure<AbpBundlingOptions>(options =>
{
    options.StyleBundles.Configure(
        StandardBundles.Styles.Global, //The bundle name!
        bundleConfiguration =>
    {
        bundleConfiguration.AddFiles("/styles/my-global-styles.css");
    }
);
````
```

### #### The Global Script Bundle

Just like the `StandardBundles.Styles.Global`, there is a `StandardBundles.Scripts.Global` that you can add files or manipulate the existing ones.

#### #### Example: Manipulate the Bundle Files

The example above adds a new file to the bundle. You can do more if you create a **\*\*bundle contributor\*\*** class. Example:

```
~~~~csharp
public class MyGlobalStyleBundleContributor : BundleContributor
{
 public override void ConfigureBundle(BundleConfigurationContext context)
 {
 context.Files.Clear();
 context.Files.Add("/styles/my-global-styles.css");
 }
}
~~~~
```

Then you can add the contributor to an existing bundle:

```
~~~~csharp
Configure<AbpBundlingOptions>(options =>
{
 options.StyleBundles.Configure(
 StandardBundles.Styles.Global,
 bundleConfiguration =>
 {
 bundleConfiguration.AddContributors(typeof(MyGlobalStyleBundleContributor));
 }
);
}
~~~~
```

It is not a good idea to clear all CSS files. In a real world scenario, you can find and replace a specific file with your own file.

#### #### Example: Add a JavaScript File for a Specific Page

The examples above works with the global bundle added to the layout. What if you want to add a CSS/JavaScript file (or replace a file) for a specific page defines inside a depended module?

Assume that you want to run a **\*\*JavaScript code\*\*** once the user enters to the **\*\*Role Management\*\*** page of the Identity Module.

First, create a standard JavaScript file under the ``wwwroot``, ``Pages`` or ``Views`` folder (ABP support to add static resources inside these folders by default). We prefer the ``Pages/Identity/Roles`` folder to follow the conventions:

```
![bookstore-added-role-js-file](../../images/bookstore-added-role-js-file.png)
```

Content of the file is simple:

```
~~~~js
```

```

$(function() {
 abp.log.info('My custom role script file has been loaded!');
});
```

```

Then add this file to the bundle of the role management page:

```

```csharp
Configure<AbpBundlingOptions>(options =>
{
 options.ScriptBundles
 .Configure(
 typeof(Volo.Abp.Identity.Web.Pages.Identity.Roles.IndexModel).FullName,
 bundleConfig =>
 {
 bundleConfig.AddFiles("/Pages/Identity/Roles/my-role-script.js");
 });
});
```

```

``typeof(Volo.Abp.Identity.Web.Pages.Identity.Roles.IndexModel).FullName`` is the safe way to get the bundle name for the role management page.

> Notice that not every page defines such page bundles. They define only if needed.

In addition to adding new CSS/JavaScript file to a page, you also can replace the existing one (by defining a bundle contributor).

Layout Customization

Layouts are defined by the theme ([see the theming](Theming.md)) by design. They are not included in a downloaded application solution. In this way you can easily ****upgrade**** the theme and get new features. You can not ****directly change**** the layout code in your application unless you replace it by your own layout (will be explained in the next sections).

There are some common ways to ****customize the layout**** described in the next sections.

Menu Contributors

There are two ****standard menus**** defined by the ABP Framework:

```
![bookstore-menus-highlighted](../../images/bookstore-menus-highlighted.png)
```

- * `StandardMenus.Main`: The main menu of the application.
- * `StandardMenus.User`: The user menu (generally at the top right of the screen).

Rendering the menus is a responsibility of the theme, but ****menu items**** are determined by the modules and your application code. Just implement the `IMenuContributor` interface and ****manipulate the menu items**** in the `ConfigureMenuAsync` method.

Menu contributors are executed whenever need to render the menu. There is already a menu contributor defined in the ****application startup template****, so you can take it as an example and improve if necessary. See the [navigation menu] (Navigation-Menu.md) document for more.

Toolbar Contributors

[Toolbar system] (Toolbars.md) is used to define ****toolbars**** on the user interface. Modules (or your application) can add ****items**** to a toolbar, then the theme renders the toolbar on the ****layout****.

There is only one ****standard toolbar**** (named "Main" – defined as a constant: `StandardToolbars.Main`). For the basic theme, it is rendered as shown below:![bookstore-toolbar-highlighted](../../images/bookstore-toolbar-highlighted.png)

In the screenshot above, there are two items added to the main toolbar: Language switch component & user menu. You can add your own items here.

Example: Add a Notification Icon

In this example, we will add a ****notification (bell) icon**** to the left of the language switch item. A item in the toolbar should be a ****view component****. So, first, create a new view component in your project:

```
![bookstore-notification-view-component](../../images/bookstore-notification-view-component.png)
```

NotificationViewComponent.cs

```
```csharp
public class NotificationViewComponent : AbpViewComponent
{
 public async Task<IViewComponentResult> InvokeAsync()
 {
 return View("/Pages/Shared/Components/Notification/Default.cshtml");
 }
}
```
```

Default.cshtml

```
```xml
<div id="MainNotificationIcon" style="color: white; margin: 8px;">
 <i class="far fa-bell"></i>
</div>
```
```

Now, we can create a class implementing the `IToolbarContributor` interface:

```
```csharp
public class MyToolbarContributor : IToolbarContributor
{
```

```

public Task ConfigureToolbarAsync(IToolbarConfigurationContext context)
{
 if (context.Toolbar.Name == StandardToolbars.Main)
 {
 context.Toolbar.Items
 .Insert(0, new ToolbarItem(typeof(NotificationViewComponent)));
 }

 return Task.CompletedTask;
}
```

```

This class adds the `NotificationViewComponent` as the first item in the `Main` toolbar.

Finally, you need to add this contributor to the `AbpToolbarOptions`, in the `ConfigureServices` of your module:

```

```csharp
Configure<AbpToolbarOptions>(options =>
{
 options CONTRIBUTORS.Add(new MyToolbarContributor());
});
```

```

That's all, you will see the notification icon on the toolbar when you run the application:

![bookstore-notification-icon-on-toolbar](../../images/bookstore-notification-icon-on-toolbar.png)

`NotificationViewComponent` in this sample simply returns a view without any data. In real life, you probably want to **query database** (or call an HTTP API) to get notifications and pass to the view. If you need, you can add a `JavaScript` or `CSS` file to the global bundle (as described before) for your toolbar item.

See the [\[toolbars document\]](#)(Toolbars.md) for more about the toolbar system.

Layout Hooks

[\[Layout Hooks\]](#)(Layout-Hooks.md) system allows you to **add code** at some specific parts of the layout. All layouts of all themes should implement these hooks. Then you can then add a **view component** into a hook point.

Example: Add Google Analytics Script

Assume that you need to add the Google Analytics script to the layout (that will be available for all the pages). First, **create a view component** in your project:

![bookstore-google-analytics-view-component](../../images/bookstore-google-analytics-view-component.png)

****GoogleAnalyticsViewComponent.cs****

```
~~~~csharp
public class GoogleAnalyticsViewComponent : AbpViewComponent
{
    public IViewComponentResult Invoke()
    {
        return View("/Pages/Shared/Components/GoogleAnalytics/Default.cshtml");
    }
}~~~
```

****Default.cshtml****

```
~~~~html
<script>
    (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
        (i[r].q=i[r].q||[]).push(arguments)},i[r].l=1*new Date();a=s.createElement(o),
m=s.getElementsByTagName(o)[0];a.async=1;a.src=g;m.parentNode.insertBefore(a,m)
}) (window,document,'script','//www.google-analytics.com/analytics.js','ga');

    ga('create', 'UA-xxxxxx-1', 'auto');
    ga('send', 'pageview');
</script>
~~~
```

Change `UA-xxxxxx-1` with your own code.

You can then add this component to any of the hook points in the `ConfigureServices` of your module:

```
~~~~csharp
Configure<AbpLayoutHookOptions>(options =>
{
    options.Add(
        LayoutHooks.Head.Last, //The hook name
        typeof(GoogleAnalyticsViewComponent) //The component to add
    );
});~~~
```

Now, the GA code will be inserted in the `head` of the page as the last item. You (or the modules you are using) can add multiple items to the same hook. All of them will be added to the layout.

The configuration above adds the `GoogleAnalyticsViewComponent` to all layouts. You may want to only add to a specific layout:

```
~~~~csharp
Configure<AbpLayoutHookOptions>(options =>
{~~~
```

```

        options.Add(
            LayoutHooks.Head.Last,
            typeof(GoogleAnalyticsViewComponent),
            layout: StandardLayouts.Application //Set the layout to add
        );
    });
```

```

See the layouts section below to learn more about the layout system.

#### #### Layouts

Layout system allows themes to define standard, named layouts and allows any page to select a proper layout for its purpose. There are three pre-defined layouts:

- \* **\*\*Application\*\***: The main (and the default) layout for an application. It typically contains header, menu (sidebar), footer, toolbar... etc.
- \* **\*\*Account\*\***: This layout is used by login, register and other similar pages. It is used for the pages under the `/Pages/Account` folder by default.
- \* **\*\*Empty\*\***: Empty and minimal layout.

These names are defined in the `StandardLayouts` class as constants. You can definitely create your own layouts, but these are the standard layout names and implemented by all the themes out of the box.

#### #### Layout Location

You can find the layout files [here](<https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Mvc.UI.Theme.Basic/Themes/Basic/Layouts>) for the basic theme. You can take them as references to build your own layouts or you can override them if necessary.

#### #### ITheme

ABP Framework uses the `ITheme` service to get the layout location by the layout name. You can replace this service to dynamically select the layout location.

#### #### IThemeManager

`IThemeManager` is used to obtain the current theme and get the layout path. Any page can determine the layout of its own. Example:

```

```html
@using Volo.Abp.AspNetCore.Mvc.UI.Theming
@inject IThemeManager ThemeManager
@{
    Layout = ThemeManager.CurrentTheme.GetLayout(StandardLayouts.Empty);
}
```

```

This page will use the empty layout. You use ``ThemeManager.CurrentTheme.GetEmptyLayout();`` extension method as a shortcut.

If you want to set the layout for all the pages under a specific folder, then write the code above in a ``_ViewStart.cshtml`` file under that folder.

#### 10.1.21.2 Entity Action Extensions

```
Entity Action Extensions for ASP.NET Core UI
```

##### ## Introduction

Entity action extension system allows you to add a **\*\*new action\*\*** to the action menu for an entity. A **\*\*Click Me\*\*** action was added to the *\*User Management\** page below:

```
![user-action-extension-click-me](../../images/user-action-extension-click-me.png)
```

You can take any action (open a modal, make an HTTP API call, redirect to another page... etc) by writing your custom code. You can access to the current entity in your code.

##### ## How to Set Up

In this example, we will add a "Click Me!" action and execute a JavaScript code for the user management page of the `[Identity Module](../../Modules/Identity.md)`.

##### ### Create a JavaScript File

First, add a new JavaScript file to your solution. We added inside the ``/Pages/Identity/Users`` folder of the ``.Web`` project:

```
![user-action-extension-on-solution](../../images/user-action-extension-on-solution.png)
```

Here, the content of this JavaScript file:

```
```js
var clickMeAction = {
    text: 'Click Me!',
    action: function(data) {
        //TODO: Write your custom code
        alert(data.record.userName);
    }
};

abp.ui.extensions.entityActions
    .get('identity.user')
    .addContributor(function(actionList) {
        actionList.addTail(clickMeAction);
    });
```

```

In the `action` function, you can do anything you need. See the API section for a detailed usage.

#### ### Add the File to the User Management Page

Then you need to add this JavaScript file to the user management page. You can take the power of the [Bundling & Minification System] (Bundling-Minification.md).

Write the following code inside the `ConfigureServices` of your module class:

```
```csharp
Configure<AbpBundlingOptions>(options =>
{
    options.ScriptBundles.Configure(
        typeof(Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel).FullName,
        bundleConfiguration =>
    {
        bundleConfiguration.AddFiles(
            "/Pages/Identity/Users/my-user-extensions.js"
        );
    });
});
```

This configuration adds `my-user-extensions.js` to the user management page of the Identity Module. `typeof(Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel).FullName` is the name of the bundle in the user management page. This is a common convention used for all the ABP Commercial modules.

That's all. Run your application to see the result.

API

This section explains details of the `abp.ui.extensions.entityActions` JavaScript API.

abp.ui.extensions.entityActions.get(entityName)

This method is used to access the entity actions of a specific module. It takes one parameter:

* **entityName**: The name of the entity defined by the related module.

abp.ui.extensions.entityActions.get(entityName).actions

The `actions` property is used to retrieve a [doubly linked list] (./Common/Utils/Linked-List.md) of previously defined actions for an entity. All contributors are executed in order to prepare the final actions list. This is normally called by the modules to show the actions in the grid. However, you can use it if you are building your own extensible UIs.

abp.ui.extensions.entityActions.get(entityName).addContributor(contributeCallback)

The `addContributor` method covers all scenarios, e.g. you want to add your action in a different position in the list, change or remove an existing action item. `addContributor` with the following parameter:

* **contributeCallback**: A callback function that is called whenever the action list should be created. You can freely modify the action list inside this callback method.

Example

```
```js
var clickMe2Action = {
 text: 'Click Me 2!',
 icon: 'fas fa-hand-point-right',
 action: function(data) {
 //TODO: Write your custom code
 alert(data.record.userName);
 }
};

abp.ui.extensions.entityActions
 .get('identity.user')
 .addContributor(function(actionList) {
 // Remove an item from actionList
 actionList.dropHead();

 // Add the new item to the actionList
 actionList.addHead(clickMe2Action);
 });
```

```

> `actionList` is [linked list](../Common/Utils/Linked-List.md). You can use its methods to build a list of columns however you need.

10.1.21.3 Data Table Column Extensions

Data Table Column Extensions for ASP.NET Core UI

Introduction

Data table column extension system allows you to add a **new table column** on the user interface. The example below adds a new column with the "Social security no" title:

```
![user-action-extension-click-me](../../images/table-column-extension-example.png)
```

You can use the standard column options to fine control the table column.

> Note that this is a low level API to find control the table column. If you want to show an extension property on the table, see the [module entity extension](../../Module-Entity-Extensions.md) document.

How to Set Up

Create a JavaScript File

First, add a new JavaScript file to your solution. We added inside the `~/Pages/Identity/Users` folder of the `~.Web` project:

```
![user-action-extension-on-solution](../../images/user-action-extension-on-solution.png)
```

Here, the content of this JavaScript file:

```
```js
abp.ui.extensions.tableColumns
 .get('identity.user')
 .addContributor(function (columnList) {
 columnList.addTail({ //add as the last column
 title: 'Social security no',
 data: 'extraProperties.SocialSecurityNumber',
 orderable: false,
 render: function (data, type, row) {
 if (row.extraProperties.SocialSecurityNumber) {
 return '' +
 row.extraProperties.SocialSecurityNumber +
 '';
 } else {
 return '<i class="text-muted">undefined</i>';
 }
 }
 });
 });
```

```

This example defines a custom `render` function to return a custom HTML to render in the column.

Add the File to the User Management Page

Then you need to add this JavaScript file to the user management page. You can take the power of the [Bundling & Minification system](<https://docs.abp.io/en/abp/latest/UI/AspNetCore/Bundling-Minification>).

Write the following code inside the `ConfigureServices` of your module class:

```
```csharp
Configure<AbpBundlingOptions>(options =>
{
 options.ScriptBundles.Configure(
 typeof(Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel).FullName,
 bundleConfiguration =>
 {
 bundleConfiguration.AddFiles(
 "/Pages/Identity/Users/my-user-extensions.js"
);
 });
});
```

```

```

        );
    });
});
```

```

This configuration adds `my-user-extensions.js` to the user management page of the Identity Module. `typeof(Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel).FullName` is the name of the bundle in the user management page. This is a common convention used for all the ABP Commercial modules.

#### #### Rendering the Column

This example assumes that you've defined a `SocialSecurityNumber` extra property using the [module entity extension](../../Module-Entity-Extensions.md) system. However;

- \* You can add a new column that is related to an existing property of the user (that was not added to the table by default). Example:

```

```js
abp.ui.extensions.tableColumns
    .get('identity.user')
    .addContributor(function (columnList) {
        columnList.addTail({
            title: 'Phone confirmed?',
            data: 'phoneNumberConfirmed',
            render: function (data, type, row) {
                if (row.phoneNumberConfirmed) {
                    return '<strong style="color: green">YES</strong>';
                } else {
                    return '<i class="text-muted">NO</i>';
                }
            }
        });
```

```

- \* You can add a new custom column that is not related to any entity property, but a completely custom information. Example:

```

```js
abp.ui.extensions.tableColumns
    .get('identity.user')
    .addContributor(function (columnList) {
        columnList.addTail({
            title: 'Custom column',
            data: {},
            orderable: false,
            render: function (data) {
                if (data.phoneNumber) {
                    return "call: " + data.phoneNumber;
                } else {
                    return '';
                }
            }
        });
```

```

```
 }
 });
....});
```

## ## API

This section explains details of the `abp.ui.extensions.tableColumns` JavaScript API.

### ### abp.ui.extensions.tableColumns.get(entityName)

This method is used to access the table columns for an entity of a specific module. It takes one parameter:

\* **entityName**: The name of the entity defined by the related module.

### ### abp.ui.extensions.tableColumns.get(entityName).columns

The `columns` property is used to retrieve a [doubly linked list](../Common/Utils/LinkedList.md) of previously defined columns for a table. All contributors are executed in order to prepare the final column list. This is normally called by the modules to show the columns in the table. However, you can use it if you are building your own extensible UIs.

### ### abp.ui.extensions.tableColumns.get(entityName).addContributor(contributeCallback [, order])

The `addContributor` method covers all scenarios, e.g. you want to add your column in a different position in the list, change or remove an existing column. `addContributor` has the following parameters:

\* **contributeCallback**: A callback function that is called whenever the column list should be created. You can freely modify the column list inside this callback method.  
\* **order** (optional): The order of the callback in the callback list. Your callback is added to the end of the list (so, you have opportunity to modify columns added by the previous contributors). You can set it `0` to add your contributor as the first item.

## #### Example

```
```js
var myColumnDefinition = {
    title: 'Custom column',
    data: {},
    orderable: false,
    render: function(data) {
        if (data.phoneNumber) {
            return "call: " + data.phoneNumber;
        } else {
            return '';
        }
    }
};
```

```

abp.ui.extensions.tableColumns
    .get('identity.user')
    .addContributor(function (columnList) {
        // Remove an item from actionList
        columnList.dropHead();

        // Add a new item to the actionList
        columnList.addHead(myColumnDefinition);
    });
```

```

> `columnList` is [linked list](../Common/Utils/Linked-List.md). You can use its methods to build a list of columns however you need.

#### 10.1.21.4 Page Toolbar Extensions

```
Page Toolbar Extensions for ASP.NET Core UI
```

Page toolbar system allows you to add components to the toolbar of any page. The page toolbar is the area right to the header of a page. A button ("Import users from excel") was added to the user management page below:

```
![page-toolbar-button](../../images/page-toolbar-button.png)
```

You can add any type of view component item to the page toolbar or modify existing items.

#### ## How to Set Up

In this example, we will add an "Import users from excel" button and execute a JavaScript code for the user management page of the [Identity Module](../../Modules/Identity.md).

#### ### Add a New Button to the User Management Page

Write the following code inside the `ConfigureServices` of your web module class:

```
```csharp
Configure<AbpPageToolbarOptions>(options =>
{
    options.Configure<Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel>(toolbar =>
    {
        toolbar.AddButton(
            LocalizableString.Create<MyProjectNameResource>("ImportFromExcel"),
            icon: "file-import",
            id: "ImportUsersFromExcel",
            type: AbpButtonType.Secondary
        );
    });
});```

```

`AddButton` is a shortcut to simply add a button component. Note that you need to add the `ImportFromExcel` to your localization dictionary (json file) to localize the text.

When you run the application, you will see the button added next to the current button list. There are some other parameters of the `AddButton` method (for example, use `order` to set the order of the button component relative to the other components).

Create a JavaScript File

Now, we can go to the client side to handle click event of the new button. First, add a new JavaScript file to your solution. We added inside the `/Pages/Identity/Users` folder of the `.Web` project:

```
![user-action-extension-on-solution](../../images/user-action-extension-on-solution.png)
```

Here, the content of this JavaScript file:

```
```js
$(function () {
 $('#ImportUsersFromExcel').click(function (e) {
 e.preventDefault();
 alert('TODO: import users from excel');
 });
});
```

In the `click` event, you can do anything you need to do.

### ### Add the File to the User Management Page

Then you need to add this JavaScript file to the user management page. You can take the power of the [Bundling & Minification system] (Bundling-Minification.md).

Write the following code inside the `ConfigureServices` of your module class:

```
```csharp
Configure<AbpBundlingOptions>(options =>
{
    options.ScriptBundles.Configure(
        typeof(Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel).FullName,
        bundleConfiguration =>
    {
        bundleConfiguration.AddFiles(
            "/Pages/Identity/Users/my-user-extensions.js"
        );
    });
});
```

This configuration adds `my-user-extensions.js` to the user management page of the Identity Module. `typeof(Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel).FullName`

is the name of the bundle in the user management page. This is a common convention used for all the ABP Commercial modules.

Advanced Use Cases

While you typically want to add a button action to the page toolbar, it is possible to add any type of component.

Add View Component to a Page Toolbar

First, create a new view component in your project:

```
![page=toolbar=custom=component](../../images/page-toolbar-custom-component.png)
```

For this example, we've created a `MyToolbarItem` view component under the `/Pages/Identity/Users/MyToolbarItem` folder.

`MyToolbarItemViewComponent.cs` content:

```
```csharp
public class MyToolbarItemViewComponent : AbpViewComponent
{
 public IViewComponentResult Invoke()
 {
 return View("~/Pages/Identity/Users/MyToolbarItem/Default.cshtml");
 }
}```

```

`Default.cshtml` content:

```
```xml
<span>
    <button type="button" class="btn btn-dark">CLICK ME</button>
</span>
```
```

- \* `\*.cshtml` file can contain any type of component(s). It is a typical view component.
- \* `MyToolbarItemViewComponent` can inject and use any service if you need.

Then you can add the `MyToolbarItemViewComponent` to the user management page:

```
```csharp
Configure<AbpPageToolbarOptions>(options =>
{
    options.Configure<Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel>(
        toolbar =>
        {
            toolbar.AddComponent<MyToolbarItemViewComponent>();
        }
    );
});```

```

```
```
```

\* If your component accepts arguments (in the `Invoke`/`InvokeAsync` method), you can pass them to the `AddComponent` method as an anonymous object.

#### #### Permissions

If your button/component should be available based on a [permission/policy](../../../../Authorization.md), you can pass the permission/policy name as the `requiredPolicyName` parameter to the `AddButton` and `AddComponent` methods.

#### ### Add a Page Toolbar Contributor

If you perform advanced custom logic while adding an item to a page toolbar, you can create a class that implements the `IPageToolbarContributor` interface or inherits from the `PageToolbarContributor` class:

```
```csharp
public class MyToolbarContributor : PageToolbarContributor
{
    public override Task ContributeAsync(PageToolbarContributionContext context)
    {
        context.Items.Insert(0, new PageToolbarItem(typeof(MyToolbarItemViewComponent)));

        return Task.CompletedTask;
    }
}
```
```

\* You can use `context.ServiceProvider` to resolve dependencies if you need.

Then add your class to the `Contributors` list:

```
```csharp
Configure<AbpPageToolbarOptions>(options =>
{
    options.Configure<Volo.Abp.Identity.Web.Pages.Identity.Users.IndexModel>(
        toolbar =>
    {
        toolbar CONTRIBUTORS.Add(new MyToolbarContributor());
    }
);
}
```
```

## 10.1.22 Security

### 10.1.22.1 Security Headers

```
Security Headers
```

ABP Framework allows you to add frequently used security headers into your application. The following security headers will be added as response headers to your application if you use the `UseAbpSecurityHeaders` middleware:

- \* `X-Content-Type-Options`: Tells the browser to not try and guess what a mime-type of a resource might be, and to just take what mime-type the server has returned.
- \* `X-XSS-Protection`: This is a feature of Internet Explorer, Chrome, and Safari that stops pages from loading when they detect reflected cross-site scripting (XSS) attacks.
- \* `X-Frame-Options`: This header can be used to indicate whether or not a browser should be allowed to render a page in a `<iframe>` tag. By specifying this header value as `*SAMEORIGIN*`, you can make it displayed in a frame on the same origin as the page itself.
- \* `Content-Security-Policy`: This response header allows you to restrict which resources (such as JavaScript, CSS, images, manifests, etc.) can be loaded, and the URLs that they can be loaded from. This security header will only be added if you configure the `AbpSecurityHeadersOptions` class and enable it.

## ## Configuration

### ### AbpSecurityHeadersOptions

`AbpSecurityHeadersOptions` is the main class to enable the `Content-Security-Policy` header, define its value and set other security headers that you want to add to your application.

#### **\*\*Example:\*\***

```
```csharp
Configure<AbpSecurityHeadersOptions>(options =>
{
    options.UseContentSecurityPolicyHeader = true; //false by default
    options.ContentSecurityPolicyValue = "object-src 'none'; form-action 'self'; frame-
ancestors 'none'"; //default value

    //adding additional security headers
    options.Headers["Referrer-Policy"] = "no-referrer";
});
```

```

➤ If the header is the same, the additional security headers you defined take precedence over the default security headers. In other words, it overrides the default security headers' values.

## ## Security Headers Middleware

Security Headers middleware is an ASP.NET Core request pipeline [middleware] (<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware>) that adds pre-defined security headers to your application, including `X-Content-Type-Options`, `X-XSS-Protection`, and `X-Frame-Options`. Additionally, this middleware also includes those unique security headers in your application if you configure the `AbpSecurityHeadersOptions` as mentioned above.

## **\*\*Example:\*\***

```
```csharp
app.UseAbpSecurityHeaders();
````
```

➤ You can add this middleware into the `OnApplicationInitialization` method of your module class to register it to the request pipeline. This middleware is already configured in the [ABP Commercial Startup Templates](<https://docs.abp.io/en/commercial/latest/startup-templates/index>), so you don't need to manually add it if you are using one of these startup templates.

After that, you have registered the `UseAbpSecurityHeaders` middleware into the request pipeline, the defined security headers will be shown in the response headers as in the figure below:



### **## Content Security Policy Script Nonce**

Abp Framework provides a property to add a dynamic script-src nonce value to the Content-Security-Policy header. With this feature, it automatically adds a dynamic nonce value to the header side. And with the help of the script tag helper, it adds this [`script nonce`]([https://developer.mozilla.org/en-US/docs/Web/HTML/Global\\_attributes/nonce](https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/nonce)) value to the script tags on your pages(The `ScriptNonceTagHelper` in the `Volo.Abp.AspNetCore.Mvc.UI.Bundling` namespace must be attached as a taghelper.).

➤ If you need to add the nonce script manually, you can use `Html.GetScriptNonce()` to add the nonce value or `Html.GetScriptNonceAttribute()` to add the nonce attribute value.

This feature is disabled by default. You can enable it by setting the `UseContentSecurityPolicyScriptNonce` property of the `AbpSecurityHeadersOptions` class to `true`.

### **### Ignore Script Nonce**

You can ignore the script nonce for some pages or some selectors. You can use the `IgnoredScriptNoncePaths` and `IgnoredScriptNonceSelectors` properties of the `AbpSecurityHeadersOptions` class.

## **\*\*Example:\*\***

```
```csharp
Configure<AbpSecurityHeadersOptions>(options =>
{
    //adding script-src nonce
    options.UseContentSecurityPolicyScriptNonce = true; //false by default

    //ignore script nonce source for these paths
    options.IgnoredScriptNoncePaths.Add("/my-page");

    //ignore script nonce by Elsa Workflows and other selectors
    options.IgnoredScriptNonceSelectors.Add(context =>
```

```

    {
        var endpoint = context.GetEndpoint();
        return
    Task.FromResult(endpoint?.Metadata.GetMetadata<PageRouteMetadata>()?.RouteTemplate ==
    "/{YOURHOSTPAGE}");
    });
}
```

```

#### ### Ignore Abp Security Headers

You can ignore the Abp Security Headers for some actions or pages. You can use the `IgnoreAbpSecurityHeaderAttribute` attribute for this.

#### **\*\*Example:\*\***

```

```csharp
@using Volo.Abp.AspNetCore.Security
@attribute [IgnoreAbpSecurityHeaderAttribute]
```

```

#### **\*\*Example:\*\***

```

```csharp
[IgnoreAbpSecurityHeaderAttribute]
public class IndexModel : AbpPageModel
{
    public void OnGet()
    {
    }
}
```

```

## 10.2 Blazor

### 10.2.1 Overall

# Blazor UI: Overall

## Introduction

[Blazor] (<https://docs.microsoft.com/en-us/aspnet/core/blazor/>) is a framework for building interactive client-side web UI with .NET. It is promising for a .NET developer that you can create Single-Page Web Applications using C# and the Razor syntax.

ABP provides infrastructure and integrations that make your Blazor development even easier, comfortable and enjoyable.

This document provides an overview for the ABP Framework Blazor UI integration and highlights some major features.

### Getting Started

You can follow the documents below to start with the ABP Framework and the Blazor UI now:

- \* [Get started](../../../../Getting-Started.md) with the Blazor UI for the ABP Framework.
- \* [Web Application Development Tutorial](../../../../Tutorials/Part-1.md) with the Blazor UI.

## ## Modularity

[Modularity](../../../../Module-Development-Basics.md) is one of the key goals of the ABP Framework. It is not different for the UI; It is possible to develop modular applications and reusable application modules with isolated and reusable UI pages and components.

The [application startup template](../../../../Startup-Templates/Application.md) comes with some application modules pre-installed. These modules have their own UI pages embedded into their own NuGet packages. You don't see their code in your solution, but they work as expected on runtime.

## ## Dynamic C# Client Proxies

Dynamic C# Client Proxy system makes extremely easy to consume server side HTTP APIs from the UI. You just **\*\*inject\*\*** the [application service](../../../../Application-Services.md) **\*\*interface\*\*** and consume the remote APIs just like using local service method calls.

**\*\*Example: Get list of books from server and list on the UI\*\***

```
~~~~csharp
@page "/books"
@using Acme.BookStore.Books
@using Volo.Abp.Application.Dtos
@inject IBookAppService BookAppService

<ul>
    @foreach (var book in Books)
    {
        <li>
            @book.Name (by @book.AuthorName)
        </li>
    }
</ul>

@code {
    private IReadOnlyList<BookDto> Books { get; set; } = new List<BookDto>();

    protected override async Task OnInitializedAsync()
    {
        var result = await BookAppService.GetListAsync(
            new PagedAndSortedResultRequestDto()
        );

        Books = result.Items;
    }
}
```

~~~~

- * This razor component (page) uses `@inject IBookAppService BookAppService` to get a reference to the service proxy.
- * It uses `BookAppService.GetListAsync` in the `OnInitializedAsync` and gets the list of the books, just like a regular C# method call.
- * Finally, the page renders the books in a list on the UI.

ABP Framework handles all the low level details for you, including a proper HTTP call, JSON serialization, exception handling and authentication.

See the [\[Dynamic C# Client Proxies\]](#)(../../API/Dynamic-CSharp-API-Clients.md) document for more.

Theming System

ABP Framework provides a complete [\[Theming\]](#)(Theming.md) system with the following goals:

- * Reusable [\[application modules\]](#)(../../Modules/Index.md) are developed ****theme-independent****, so they can work with any UI theme.
- * UI theme is ****decided by the final application****.
- * The theme is distributed as NuGet package, so it is ****easily upgradable****.
- * The final application can ****customize**** the selected theme.

Current Themes

Currently, three themes are ****officially provided****:

- * The [\[Basic Theme\]](#)(Basic-Theme.md) is the minimalist theme with the plain Bootstrap style. It is ****open source and free****.
- * The [\[Lepton Theme\]](#)(<https://commercial.abp.io/themes>) is a ****commercial**** theme developed by the core ABP team and is a part of the [\[ABP Commercial\]](#)(<https://commercial.abp.io/>) license.
- * The [\[LeptonX Theme\]](#)(<https://x.leptontheme.com/>) is a theme that has both [\[commercial\]](#)(<https://docs.abp.io/en/commercial/latest/themes/lepton-x/commercial/blazor>) and [\[lite\]](#)(../../Themes/LeptonXLite/Blazor.md) choices.

Base Libraries

There are a set of standard libraries that comes pre-installed and supported by all the themes:

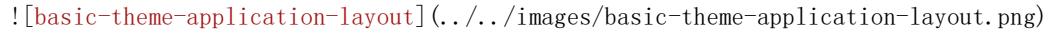
- * [\[Twitter Bootstrap\]](#)(<https://getbootstrap.com/>) as the fundamental HTML/CSS framework.
- * [\[Blazorise\]](#)(<https://github.com/stsrki/Blazorise>) as a component library that supports the Bootstrap and adds extra components like Data Grid and Tree.
- * [\[FontAwesome\]](#)(<https://fontawesome.com/>) as the fundamental CSS font library.
- * [\[Flag Icon\]](#)(<https://github.com/lipis/iconflag>) as a library to show flags of countries.

These libraries are selected as the base libraries and available to the applications and modules.

- Bootstrap's JavaScript part is not used since the Blazorise library already provides the necessary functionalities to the Bootstrap components in a native way.
- Beginning from June, 2021, the Blazorise library has dual licenses; open source & commercial. Based on your yearly revenue, you may need to buy a commercial license. See [[this post](https://blazorise.com/news/announcing-2022-blazorise-plans-and-pricing-updates)] (<https://blazorise.com/news/announcing-2022-blazorise-plans-and-pricing-updates>) to learn more. The Blazorise license is bundled with ABP Commercial and commercial customers doesn't need to buy an extra Blazorise license.

The Layout

The themes provide the layout. So, you have a responsive layout with the standard features already implemented. The screenshot below has taken from the layout of the [[Basic Theme](#)] ([Basic-Theme.md](#)) :

 ![basic-theme-application-layout] (../../images/basic-theme-application-layout.png)

See the [[Theming](#)] ([Theming.md](#)) document for more layout options and other details.

Layout Parts

A typical layout consists of multiple parts. The [[Theming](#)] ([Theming.md](#)) system provides [[menus](#)] ([Navigation-Menu.md](#)), [[toolbars](#)] ([Toolbars.md](#)), [[page alerts](#)] ([Page-Alerts.md](#)) and more to dynamically control the layout by your application and the modules you are using.

Global Styles & Scripts / Bundling & Minification

ABP provides a standard way to manage the global script and style dependencies of an application. This is an essential feature for modularity since some modules may have such dependencies and they can declare dependencies in that way.

See the [[Managing Global Scripts & Styles](#)] ([Global-Scripts-Styles.md](#)) document.

Services

ABP provides useful services that you can consume in your applications. Some of them are;

- * [[IUiMessageService](#)] ([Message.md](#)) is used to show modal messages to the user.
- * [[IUiNotificationService](#)] ([Notification.md](#)) is used to show toast-style notifications.
- * [[IAlertManager](#)] ([Page-Alerts.md](#)) is used to show in-page alerts.
- * [[ISettingProvider](#)] ([Settings.md](#)) is used to access to the current setting values.
- * `ICurrentUser` and `ICurrentTenant` is used to get information about the current user and the tenant.

Dependency Injection

Razor components doesn't support [[constructor injection](#)] (../../Dependency-Injection.md) by default. ABP makes possible to inject dependencies into the constructor of the code-behind file of a component.

****Example: Constructor-inject a service in the code-behind file of a component****

```

```csharp
using Microsoft.AspNetCore.Components;

namespace MyProject.Blazor.Pages
{
 public partial class Index
 {
 private readonly NavigationManager _navigationManager;

 public Index(NavigationManager navigationManager)
 {
 _navigationManager = navigationManager;
 }
 }
}
```

```

ABP makes this possible by auto registering components to and resolving the component from the [\[Dependency Injection\]](#) (./../Dependency-Injection.md) system.

➤ You can still continue to use property injection and the standard `#[Inject]` approach if you prefer.

Resolving a component from the Dependency Injection system makes it possible to easily replace components of a depended module.

Error Handling

Blazor, by default, shows a yellow line at the bottom of the page if any unhandled exception occurs. However, this is not useful in a real application.

ABP provides an [\[automatic error handling system\]](#) (Error-Handling.md) for the Blazor UI.

Customization

While the theme and some modules come as NuGet packages, you can still replace/override and customize them on need. See the [\[Customization / Overriding Components\]](#) (Customization-Overriding-Components.md) document.

10.2.2 Navigation / Menu

Blazor UI: Navigation / Menu

Every application has a main menu to allow users to navigate to pages/screens of the application. Some applications may contain more than one menu in different sections of the UI.

ABP Framework is a [\[modular\]](#) (./../Module-Development-Basics.md) application development framework. ****Every module may need to add items to the menu**.**

So, ABP Framework **provides a menu infrastructure** where;

- * The application or the modules can add items to a menu, without knowing how the menu is rendered.
- * The [theme] (Theming.md) properly renders the menu.

Adding Menu Items

In order to add menu items (or manipulate the existing items) you need to create a class implementing the `IMenuContributor` interface.

> The [application startup template](../../../../Startup-Templates/Application.md) already contains an implementation of the `IMenuContributor`. So, you can add items inside that class instead of creating a new one.

Example: Add a *CRM* menu item with *Customers* and *Orders* sub menu items**

```
```csharp
using System.Threading.Tasks;
using MyProject.Localization;
using Volo.Abp.UI.Navigation;

namespace MyProject.Web.Menus
{
 public class MyProjectMenuContributor : IMenuContributor
 {
 public async Task ConfigureMenuAsync(MenuConfigurationContext context)
 {
 if (context.Menu.Name == StandardMenus.Main)
 {
 await ConfigureMainMenuAsync(context);
 }
 }

 private async Task ConfigureMainMenuAsync(MenuConfigurationContext context)
 {
 var l = context.GetLocalizer<MyProjectResource>();

 context.Menu.AddItem(
 new ApplicationMenuItem("MyProject.Crm", l["Menu:CRM"])
 .AddItem(new ApplicationMenuItem(
 name: "MyProject.Crm.Customers",
 displayName: l["Menu:Customers"],
 url: "/crm/customers"))
 .AddItem(new ApplicationMenuItem(
 name: "MyProject.Crm.Orders",
 displayName: l["Menu:Orders"],
 url: "/crm/orders"))
);
 }
 }
}
```

```
}
```

\* This example adds items only to the main menu (``StandardMenus.Main`` : see the *\*Standard Menus\** section below).

\* It gets a ``IStringLocalizer`` from `context` to [localize](../../Localization.md) the display names of the menu items.

\* Adds the Customers and Orders as children of the CRM menu.

Once you create a menu contributor, you need to add it to the ``AbpNavigationOptions`` in the `ConfigureServices` method of your module:

```
```csharp
Configure<AbpNavigationOptions>(options =>
{
    options.MenuContributors.Add(new MyProjectMenuContributor());
});
````
```

This example uses some localization keys as display names those should be defined in the localization file:

```
```json
"Menu:CRM": "CRM",
"Menu:Orders": "Orders",
"Menu:Customers": "Customers"
````
```

See the [localization document](../../Localization.md) to learn more about the localization.

When you run the application, you will see the menu items added to the main menu:

```
![nav=main-menu](../../images/nav-main-menu.png)
```

> The menu is rendered by the current UI [theme](Theming.md). So, the look of the main menu can be completely different based on your theme.

Here, a few notes on the menu contributors;

- \* ABP Framework calls the `ConfigureMenuAsync` method **\*\*whenever need to render\*\*** the menu.
- \* Every menu item can have **\*\*children\*\***. So, you can add menu items with **\*\*unlimited depth\*\*** (however, your UI theme may not support unlimited depth).
- \* Only leaf menu items have `url`'s normally. When you click to a parent menu, its sub menu is opened or closed, you don't navigate the `url` of a parent menu item.
- \* If a menu item has no children and has no `url` defined, then it is not rendered on the UI. This simplifies to authorize the menu items: You only authorize the child items (see the next section). If none of the children are authorized, then the parent automatically disappears.

### Menu Item Properties

There are more options of a menu item (the constructor of the `ApplicationMenuItem` class). Here, the list of all available options;

- \* `name` (`string`, required): The \*\*unique name\*\* of the menu item.
- \* `displayName` (`string`, required): Display name/text of the menu item. You can [localize](../../Localization.md) this as shown before.
- \* `url` (`string`): The URL of the menu item.
- \* `icon` (`string`): An icon name. Free [Font Awesome](https://fontawesome.com/) icon classes are supported out of the box. Example: `fa fa-book`. You can use any CSS font icon class as long as you include the necessary CSS files to your application.
- \* `order` (`int`): The order of the menu item. Default value is `1000`. Items are sorted by the adding order unless you specify an order value.
- \* `customData` (`object`): A custom object that you can associate to the menu item and use it while rendering the menu item.
- \* `target` (`string`): Target of the menu item. Can be `null` (default), `"\_blank"`, `"\_self"`, `"\_parent"`, `"\_top"` or a frame name for web applications.
- \* `elementId` (`string`): Can be used to render the element with a specific HTML `id` attribute.
- \* `cssClass` (`string`): Additional string classes for the menu item.
- \* `groupName` (`string`): Can be used to group menu items.

#### #### Authorization

As seen above, a menu contributor contributes to the menu dynamically. So, you can perform any custom logic or get menu items from any source.

One use case is the [authorization](Authorization.md). You typically want to add menu items by checking a permission.

#### **\*\*Example: Check if the current user has a permission\*\***

```
```csharp
if (await context.IsGrantedAsync("MyPermissionName"))
{
    //...add menu items
}
```
```

> You can use `context.AuthorizationService` to directly access to the `IAuthorizationService`.

#### #### Resolving Dependencies

`context.ServiceProvider` can be used to resolve any service dependency.

#### **\*\*Example: Get a service\*\***

```
```csharp
var myService = context.ServiceProvider.GetRequiredService<IMyService>();
//...use the service
```
```

> You don't need to care about releasing/disposing services. ABP Framework handles it.

### ### The Administration Menu

There is a special menu item in the menu that is added by the ABP Framework: The **\*Administration\*** menu. It is typically used by the pre-built admin [application modules](../../Modules/Index.md) :

```
![nav=main-menu-administration](../../images/nav-main-menu-administration.png)
```

If you want to add menu items under the **\*Administration\*** menu item, you can use the ``context.Menu.GetAdministration()`` extension method:

```
~~~~csharp
context.Menu.GetAdministration().AddItem(...)
```

### ### Manipulating the Existing Menu Items

ABP Framework executes the menu contributors by the [module dependency order](../../Module-Development-Basics.md). So, you can manipulate the menu items that your application or module (directly or indirectly) depends on.

**\*\*Example: Set an icon for the `Users` menu item added by the [Identity Module](../../Modules/Identity.md)\*\***

```
~~~~csharp
var userMenu = context.Menu.FindMenuItem(IdentityMenuNames.Users);
userMenu.Icon = "fa fa-users";
```

> ``context.Menu`` gives you ability to access to all the menu items those have been added by the previous menu contributors.

### ### Menu Groups

You can define groups and associate menu items with a group.

Example:

```
~~~csharp
using System.Threading.Tasks;
using MyProject.Localization;
using Volo.Abp.UI.Navigation;

namespace MyProject.Web.Menus
{
    public class MyProjectMenuContributor : IMenuContributor
    {
        public async Task ConfigureMenuAsync(MenuConfigurationContext context)
        {
```

```
        if (context.Menu.Name == StandardMenus.Main)
        {
            await ConfigureMainMenuAsync(context);
        }
    }

private async Task ConfigureMainMenuAsync(MenuConfigurationContext context)
{
    var l = context.GetLocalizer<MyProjectResource>();

    context.Menu.AddGroup(
        new ApplicationMenuGroup(
            name: "Main",
            displayName: l["Main"]
        )
    )
    context.Menu.AddItem(
        new ApplicationMenuItem("MyProject.Crm", l["Menu:CRM"], groupName: "Main")
            .AddItem(new ApplicationMenuItem(
                name: "MyProject.Crm.Customers",
                displayName: l["Menu:Customers"],
                url: "/crm/customers")
            ).AddItem(new ApplicationMenuItem(
                name: "MyProject.Crm.Orders",
                displayName: l["Menu:Orders"],
                url: "/crm/orders")
            )
    );
}
}
```
```
```

- The UI theme will decide whether to render the groups or not, and if it decides to render, the way it's rendered is up to the theme. Only the LeptonX theme implements the menu group.

## ## Standard Menus

A menu is a **\*\*named\*\*** component. An application may contain more than one menus with different, unique names. There are two pre-defined standard menus:

- \* `Main` : The main menu of the application. Contains links to the page of the application. Defined as a constant: `Volo.Abp.UI.Navigation.StandardMenus.Main`.
  - \* `User` : User profile menu. Defined as a constant: `Volo.Abp.UI.Navigation.StandardMenus.User`.

The `Main` menu already covered above. The `User` menu is available when a user has logged in:

! [user-menu] (../../images/user-menu.png)

You can add items to the `User` menu by checking the `context.Menu.Name` as shown below:

```
```csharp
if (context.Menu.Name == StandardMenus.User)
{
    //...add items
}
```

```

## IMenuManager

`IMenuManager` is generally used by the UI [theme](Theming.md) to render the menu items on the UI. So, **you generally don't need to directly use** the `IMenuManager`.

**Example: Get the Main Menu to render in a razor component\***

```
```csharp
// Code behind file of a razor component
public partial class NavMenu
{
    private readonly IMenuManager _menuManager;

    public NavMenu(IMenuManager menuManager)
    {
        _menuManager = menuManager;
    }

    protected override async Task OnInitializedAsync()
    {
        var menu = await _menuManager.GetAsync(StandardMenus.Main);
        //...
    }
}
```

```

### 10.2.3 Localization

# Blazor UI: Localization

Blazor applications can reuse the same `IStringLocalizer<T>` service that is explained in the [localization document](../../Localization.md).

All the localization resources and texts available in the server side are usable in the Blazor application.

## IStringLocalizer

`IStringLocalizer<T>` (`T` is the localization resource class) can be injected in any service or component to use the localization service.

#### #### Razor Components

Use `@inject IStringLocalizer<MyProjectNameResource>` to use the localization in a razor component.

##### **\*\*Example: Localization in a Razor Component\*\***

```
```csharp
@page "/"
@using MyCompanyName.MyProjectName.Localization
@using Microsoft.Extensions.Localization
@inject IStringLocalizer<MyProjectNameResource> L

<h1>
    @L["LongWelcomeMessage"]
</h1>
```
```

> `L` is a name that we love and use as the name of a `IStringLocalizer` instance, while you can give any name.

#### ##### The AbpComponentBase

`AbpComponentBase` is a useful base class that you can derive the components from. It has some useful properties/methods you typically need in a component.

The `AbpComponentBase` already defines a base `L` property (of type `IStringLocalizer`). It only requires to set the resource type (in the constructor of the derived class). If you created your application from the ABP's application startup template, then you should have a \*YourProjectComponentBase\* class in the Blazor project. Inherit components from this class to have the localizer pre-injected.

##### **\*\*Example: Derive from the base component class\*\***

```
```csharp
@page "/"
@inherits MyProjectNameComponentBase

<h1>
    @L["LongWelcomeMessage"]
</h1>
```
```

#### ### Other Services

`IStringLocalizer<T>` can be injected into any service.

##### **\*\*Example\*\***

```
```csharp
public class MyService : ITransientDependency
```

```

{
    private readonly IStringLocalizer<TestResource> _localizer;

    public MyService(IStringLocalizer<TestResource> localizer)
    {
        _localizer = localizer;
    }

    public void Foo()
    {
        var str = _localizer["HelloWorld"];
    }
}
```

```

### ### Format Arguments

Format arguments can be passed after the localization key. If your message is `Hello {0}, welcome!`, then you can pass the `{0}` argument to the localizer like `\_localizer["HelloMessage", "John"]`.

> Refer to the [Microsoft's localization documentation](<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/localization>) for details about using the localization.

### ## See Also

- \* [Localization](../../Localization.md)

## 10.2.4 Theming

### 10.2.4.1 Overall

#### # Blazor UI: Theming

```

```json
//#[doc-params]
{
    "UI": ["Blazor", "BlazorServer"]
}
```

```

### ## Introduction

ABP Framework provides a complete **UI Theming** system with the following goals:

- \* Reusable [application modules](../../Modules/Index.md) are developed **theme-independent**, so they can work with any UI theme.
- \* UI theme is **decided by the final application**.
- \* The theme is distributed via a NuGet package, so it is **easily upgradable**.
- \* The final application can **customize** the selected theme.

In order to accomplish these goals, ABP Framework;

- \* Determines a set of **base libraries** used and adapted by all the themes. So, module and application developers can depend on and use these libraries without depending on a particular theme.
- \* Provides a system that consists of layout parts (like [navigation menus] (Navigation-Menu.md) and [toolbars] (Toolbars.md)) that is implemented by all the themes. So, the modules and the application to contribute to the layout to compose a consistent application UI.

### ### Current Themes

Currently, three themes are **officially provided**:

- \* The [Basic Theme] (Basic-Theme.md) is the minimalist theme with the plain Bootstrap style. It is **open source and free**.
- \* The [Lepton Theme] (<https://commercial.abp.io/themes>) is a **commercial** theme developed by the core ABP team and is a part of the [ABP Commercial] (<https://commercial.abp.io/>) license.
- \* The [LeptonX Theme] (<https://x.leptontheme.com/>) is a theme that has a [commercial] (<https://docs.abp.io/en/commercial/latest/themes/lepton-x/commercial/blazor>) and a [lite] (../../Themes/LeptonXLite/Blazor.md) version.

### ## Overall

#### ### The Base Libraries

```
{{if UI == "Blazor"}}
```

All the themes must depend on the [Volo.Abp.AspNetCore.Components.WebAssembly.Theming] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Components.WebAssembly.Theming>) NuGet package, so they are indirectly depending on the following libraries:

```
 {{end}}
```

```
 {{if UI == "BlazorServer"}}
```

All the themes must depend on the [Volo.Abp.AspNetCore.Components.Server.Theming] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Components.Server.Theming>) NuGet package, so they are indirectly depending on the following libraries:

```
 {{end}}
```

- \* [Twitter Bootstrap] (<https://getbootstrap.com/>) as the fundamental HTML/CSS framework.
- \* [Blazorise] (<https://github.com/stsrki/Blazorise>) as a component library that supports the Bootstrap and adds extra components like Data Grid and Tree.
- \* [FontAwesome] (<https://fontawesome.com/>) as the fundamental CSS font library.
- \* [Flag Icon] (<https://github.com/lipis/iconflag>) as a library to show flags of countries.

These libraries are selected as the base libraries and available to the applications and modules.

➤ Bootstrap's JavaScript part is not used since the Blazorise library already provides the necessary functionalities to the Bootstrap components in a native way.

### ### The Layout

All themes must define a layout for the application. The following image shows the user management page in the [Basic Theme] (Basic-Theme.md) application layout:

```
![basic-theme-application-layout-blazor](../../images/basic-theme-application-layout-blazor.png)
```

And the same page is shown below with the [Lepton Theme] (<https://commercial.abp.io/themes>) application layout:

```
![lepton-theme-application-layout](../../images/lepton-theme-blazor-layout.png)
```

As you can see, the page is the same, but the look is completely different in the themes above.

The application layout typically includes the following parts;

- \* A [main menu] (Navigation-Menu.md)
- \* Main [Toolbar] (Toolbars.md) with the following components;
  - \* User menu
  - \* Language switch dropdown
- \* [Page alerts] (Page-Alerts.md)
- \* The page content (aka `@Body`)

## ## Implementing a Theme

A theme is simply a Razor Class Library.

### ### The Easiest Way

The easiest way of creating a new theme is adding [Basic Theme Source Code] (<https://github.com/abpframework/abp/tree/dev/modules/basic-theme>) module with source codes and customizing it.

```
{{if UI == "Blazor"}}
```
bash
abp add-package Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme --with-source-code --
-add-to-solution-file
```
{{end}}

{{if UI == "BlazorServer"}}
```
bash
abp add-package Volo.Abp.AspNetCore.Components.Server.BasicTheme --with-source-code --add-
to-solution-file
```
```

```
```
{{end}}
```

### ### Global Styles / Scripts

A theme generally needs to add a global style to the page. ABP provides a system to manage the [Global Styles and Scripts](Global-Scripts-Styles.md). A theme can implement the `IBundleContributor` to add global style or script files to the page.

#### \*\*Example: Adding a style to the page\*\*

```
```csharp
using Volo.Abp.Bundling;

namespace MyTheme
{
    public class MyThemeBundleContributor : IBundleContributor
    {
        public void AddScripts(BundleContext context)
        {

        }

        public void AddStyles(BundleContext context)
        {
            context.Add("_content/MyTheme/styles.css");
        }
    }
}```

```

`styles.css` file should be added into the `wwwroot` folder of the theme project for this example. When you use the `abp bundle` command, this class is automatically discovered and executed to add the style to the page.

See the [Global Styles and Scripts](Global-Scripts-Styles.md) document for more.

### ### Layout Parts

A typical Layout consists of several parts. The theme should include the necessary parts in each layout.

#### \*\*Example: The Basic Theme has the following parts for the Application Layout\*\*

```
![basic-theme-application-layout-parts](../../images/basic-theme-application-layout-parts.png)
```

The application code and the modules can only show contents in the Page Content part. If they need to change the other parts (to add a menu item, to add a toolbar item, to change the application name in the branding area...) they should use the ABP Framework APIs.

The following sections explain the fundamental parts pre-defined by the ABP Framework and can be implemented by the themes.

➤ It is a good practice to split the layout into components/partials, so the final application can override them partially for customization purpose.

#### #### Branding

`IBrandingProvider` service should be used to get the name and the logo URL of the application to render in the Branding part.

The [Application Startup Template](../../Startup-Templates/Application.md) has an implementation of this interface to set the values by the application developer.

#### #### Main Menu

`IMenuManager` service is used to get the main menu items and render on the layout.

**\*\*Example: Get the Main Menu to render in a razor component\*\***

```
```csharp
// Code behind file of a razor component
public partial class NavMenu
{
    private readonly IMenuManager _menuManager;

    public NavMenu(IMenuManager menuManager)
    {
        _menuManager = menuManager;
    }

    protected override async Task OnInitializedAsync()
    {
        var menu = await _menuManager.GetAsync(StandardMenus.Main);
        //...
    }
}
```

```

See the [Navigation / Menus](Navigation-Menu.md) document to learn more about the navigation system.

#### #### Main Toolbar

`IToolbarManager` service is used to get the Main Toolbar items and render on the layout. Each item of this toolbar is a Razor Component, so it may include any type of UI elements. Inject the `IToolbarManager` and use the `GetAsync` to get the toolbar items:

```
```csharp
var toolbar = await _toolbarManager.GetAsync(StandardToolbars.Main);
```

```

› See the [Toolbars](Toolbars.md) document to learn more on the toolbar system.

The theme has a responsibility to add two pre-defined items to the main toolbar: Language Selection and User Menu. To do that, create a class implementing the `IToolbarContributor` interface and add it to the `AbpToolbarOptions` as shown below:

```
```csharp
Configure<AbpToolbarOptions>(options =>
{
    options CONTRIBUTORS.Add(new BasicThemeMainTopToolbarContributor());
});
```

```

#### ##### Language Selection

Language Selection toolbar item is generally a dropdown that is used to switch between languages. `ILanguageProvider` is used to get the list of available languages and `CultureInfo.CurrentCulture` is used to learn the current language.

```
{{if UI == "Blazor"}}
```

Local Storage is used to get and set the current language with the `Abp.SelectedLanguage` key.

#### **\*\*Example: Get the currently selected language\*\***

```
```csharp
var selectedLanguageName = await JsRuntime.InvokeAsync<string>(
    "localStorage.getItem",
    "Abp.SelectedLanguage"
);
```

```

#### **\*\*Example: Set the selected language\*\***

```
```csharp
await JsRuntime.InvokeVoidAsync(
    "localStorage.setItem",
    "Abp.SelectedLanguage",
    "en-US"
);
```

```

The theme should reload the page after changing the language:

```
```csharp
await JsRuntime.InvokeVoidAsync("location.reload");
```
}

{{end}}
```

```
 {{if UI == "BlazorServer"}}

Localization works on Server side in Blazor Server. So, regular AspNetCore localization middleware is used.
```

#### \*\*Example: Get the currently selected language\*\*

```
```csharp
var selectedLanguageName = CultureInfo.CurrentCulture.Name;
````
```

#### \*\*Example: Set the selected language\*\*

```
```csharp
// Get current url.
var relativeUrl =
    NavigationManager.Uri.RemovePreFix(NavigationManager.BaseUri).EnsureStartsWith('/');

// Redirect to ABP language switch endpoint.
NavigationManager.NavigateTo(
    $"#/Abp/Languages/Switch?culture={newLanguage.CultureName}&uiCulture={newLanguage.UICulture
    Name}&returnUrl={relativeUrl}",
    forceLoad: true
);
````

{{end}}
```

#### ##### User Menu

User menu includes links related to the user account. `IMenuManager` is used just like the Main Menu, but this time with `StandardMenus.User` parameter like shown below:

```
```csharp
var menu = await _menuManager.GetAsync(StandardMenus.User);
````
```

[ICurrentUser](../../CurrentUser.md) and [ICurrentTenant](../../Multi-Tenancy.md) services can be used to obtain the current user and tenant names.

#### #### Page Alerts

`IAlertManager` service is used to get the current page alerts to render on the layout. See the [Page Alerts](Page-Alerts.md) document to learn more.

#### 10.2.4.2 The Basic Theme

##### # Blazor UI: Basic Theme

```
```json
//[doc-params]
{
```

```
        "UI": ["Blazor", "BlazorServer"]  
    }  
}
```

The Basic Theme is a theme implementation for the Blazor UI. It is a minimalist theme that doesn't add any styling on top of the plain [Bootstrap] (<https://getbootstrap.com/>). You can take the Basic Theme as the **base theme** and build your own theme or styling on top of it. See the *Customization* section.

- If you are looking for a professional, enterprise ready theme, you can check the [Lepton Theme] (<https://commercial.abp.io/themes>), which is a part of the [ABP Commercial] (<https://commercial.abp.io/>).
- See the [Theming document] (Theming.md) to learn about themes.

## ## Installation

If you need to manually this theme, follow the steps below:

```
{{if UI == "Blazor"}}

* Install the  
[Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme] (https://www.nuget.org/packages/Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme) NuGet package to your web project.  
* Add `AbpAspNetCoreComponentsWebAssemblyBasicThemeModule` into the `[DependsOn(...)]` attribute for your [module class] (../../Module-Development-Basics.md) in the your Blazor UI project.  
* Use `Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme.Themes.Basic.App` as the root component of your application in the `ConfigureServices` method of your module:
```

```
```csharp  
var builder = context.Services.GetSingletonInstance<WebAssemblyHostBuilder>();  
builder.RootComponents.Add<App>("#ApplicationContainer");  
```\n`#ApplicationContainer` is a selector (like `

Loading...

`) in the `index.html`.
```

\* Execute `abp bundle` command under blazor project once.

```
{end}
```

```
 {{if UI == "BlazorServer"}}

* Make sure [AspNetCore Basic Theme] (../../AspNetCore/Basic-Theme.md) installation steps are completed.
```

\* Install the  
[Volo.Abp.AspNetCore.Components.Server.BasicTheme] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Components.Server.BasicTheme>) NuGet package to your web project.

```
* Add `AbpAspNetCoreComponentsServerBasicThemeModule` into the `[DependsOn(...)]` attribute for your [module class](../../Module-Development-Basics.md) in the your Blazor UI project.

* Perform following changes in `Pages/_Host.cshtml` file
  * Add usings at the top of the page.
    ```html
    @using Volo.Abp.AspNetCore.Components.Server.BasicTheme.Bundling
    @using Volo.Abp.AspNetCore.Components.Web.BasicTheme.Themes.Basic
    ```

  * Add Basic theme style bundles between `` tags.
    ```html
    <abp-style-bundle name="@BlazorBasicThemeBundles.Styles.Global" />
    ```

  * Add `App` component of Basic Theme in the body section of page.
    ```html
    <component type="typeof(App)" render-mode="Server" />
    ```

{{end}}
```

## ## The Layout

![basic-theme-application-layout](../../images/basic-theme-application-layout.png)

Application Layout implements the following parts, in addition to the common parts mentioned above;

- \* [Branding] (Branding.md) Area
- \* Main [Menu] (Navigation-Menu.md)
- \* Main [Toolbar] (Toolbars.md) with Language Selection & User Menu
- \* [Page Alerts] (Page-Alerts.md)

## ## Customization

You have two options two customize this theme:

### ### Overriding Styles / Components

In this approach, you continue to use the theme as NuGet and NPM packages and customize the parts you need to. There are several ways to customize it;

#### #### Override the Styles

You can simply override the styles in the Global Styles file of your application.

#### #### Override the Components

See the [Customization / Overriding Components] (Customization-Overriding-Components.md) to learn how you can replace components, customize and extend the user interface.

#### ### Overriding the Menu Item

Basic theme supports overriding a single menu item with a custom component. You can create a custom component and call `UseComponent` extension method of Basic Theme in the **\*\*MenuContributor\*\***.

```
```csharp
using Volo.Abp.AspNetCore.Components.Web.BasicTheme.Navigation;

//...

context.Menu.Items.Add(
    new ApplicationMenuItem("Custom.1", "My Custom Menu", "#")
        .UseComponent(typeof(MyMenuItemComponent)));
```

```html
<li class="nav-item">
    <a href="#" class="nav-link">
        My Custom Menu
    </a>
</li>
```

```

#### ### Copy & Customize

You can run the following [ABP CLI](../../CLI.md) command in **\*\*Blazor{{if UI == "Blazor"}}WebAssembly{{else}} Server{{end}}\*\*** project directory to copy the source code to your solution:

```
{{if UI == "Blazor"}}

`abp add-package Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme --with-source-code
--add-to-solution-file`
```

Then, navigate to downloaded `Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme` project directory and run:

```
`abp add-package Volo.Abp.AspNetCore.Components.Web.BasicTheme --with-source-code --add-
to-solution-file`{{end} }

{{if UI == "BlazorServer"}}

`abp add-package Volo.Abp.AspNetCore.Components.Server.BasicTheme --with-source-code --
add-to-solution-file`
```

Then, navigate to downloaded `Volo.Abp.AspNetCore.Components.Server.BasicTheme` project directory and run:

```
`abp add-package Volo.Abp.AspNetCore.Components.Web.BasicTheme --with-source-code --add-
to-solution-file`
```

```
{ {end} }
```

----

Or, you can download the source code of the Basic Theme, manually copy the project content into your solution, re-arrange the package/module dependencies (see the Installation section above to understand how it was installed to the project) and freely customize the theme based on your application requirements.

```
 {{if UI == "Blazor"}}
- [Basic Theme Source Code] (https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme)
{{end}}
```

```
 {{if UI == "BlazorServer"}}
- [Basic Theme Source Code] (https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Components.Server.BasicTheme)
{{end}}
```

## See Also

- \* [Theming] (Theming.md)

#### 10.2.4.3 LeptonX Lite

```
# LeptonX Lite Blazor UI
```

```
```json
//[doc-params]
{
  "UI": ["Blazor", "BlazorServer"]
}
````
```

LeptonX Lite has implementation for the ABP Framework Blazor WebAssembly & Blazor Server. It's a simplified variation of the [LeptonX Theme] (https://x.leptontheme.com/).

- > If you are looking for a professional, enterprise ready theme, you can check the [LeptonX Theme] (https://x.leptontheme.com/), which is a part of [ABP Commercial] (https://commercial.abp.io/).
- > See the [Theming document] (https://docs.abp.io/en/abp/latest/UI/AspNetCore/Theming) to learn about themes.

## Installation

This theme is **\*\*already installed\*\*** when you create a new solution using the startup templates. If you are using any other template, you can install this theme by following the steps below:

```
 {{if UI == "Blazor"}}
- Complete the [MVC Razor Pages Installation] (AspNetCore.md#installation) for the
**HttpApi.Host** application first. If the solution is tiered/micro-service, complete the
MVC steps for all MVC applications such as **HttpApi.Host** and if Auth Server is
separated, install to the **OpenIdict**.
```

- Add **\*\*Volo.Abp.AspNetCore.Components.WebAssembly.LeptonXLiteTheme\*\*** package to your
**\*\*Blazor WebAssembly\*\*** application with the following command:

```
```bash
dotnet add package Volo.Abp.AspNetCore.Components.WebAssembly.LeptonXLiteTheme --prerelease
```

```

- Remove **\*\*Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme\*\*** reference from the
project since it's not necessary after switching to LeptonX Lite.
- Remove the old theme from the **\*\*DependsOn\*\*** attribute in your module class and add the
**\*\*AbpAspNetCoreComponentsWebAssemblyLeptonXLiteThemeModule\*\*** type to the **\*\*DependsOn\*\***
attribute.

```
```diff
[DependsOn(
    // Remove BasicTheme module from DependsOn attribute
-    typeof(AbpAspNetCoreComponentsWebAssemblyBasicThemeModule),
    // Add LeptonX Lite module to DependsOn attribute
+    typeof(AbpAspNetCoreComponentsWebAssemblyLeptonXLiteThemeModule),
)]
```

```

- Change startup App component with the LeptonX one.

```
```csharp
// Make sure the 'App' comes from
'Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite' namespace.
builder.RootComponents.Add<App>("#ApplicationContainer");
```

```

- Run the `abp bundle` command in your **\*\*Blazor\*\*** application folder.

```
{ {end} }
```

```
 {{if UI == "BlazorServer"}}

```

- Complete the [MVC Razor Pages Installation] (AspNetCore.md#installation) first. *If the
solution is tiered/micro-service, complete the MVC steps for all MVC applications such as
**\*\*HttpApi.Host\*\*** and **\*\*AuthServer\*\***.*

- Add **`Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme`** package to your **`Blazor server`** application with the following command:

```
```bash
dotnet add package Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme --prerelease
```

```

- Remove **`Volo.Abp.AspNetCore.Components.Server.BasicTheme`** reference from the project since it's not necessary after switching to LeptonX Lite.

- Remove old theme from the **`DependsOn`** attribute in your module class and add the **`AbpAspNetCoreComponentsServerLeptonXLiteThemeModule`** type to the **`DependsOn`** attribute.

```
```diff
[DependsOn(
    // Remove BasicTheme module from DependsOn attribute
    -    typeof(AbpAspNetCoreComponentsServerBasicThemeModule),
    // Add LeptonX Lite module to DependsOn attribute
    +    typeof(AbpAspNetCoreComponentsServerLeptonXLiteThemeModule)
)]
```

```

- Replace `BlazorBasicThemeBundles` with `BlazorLeptonXLiteThemeBundles` in `AbpBundlingOptions`:
 

```
```diff
options.StyleBundles.Configure(
    // Remove following line
    - BlazorBasicThemeBundles.Styles.Global,
        // Add following line instead
    + BlazorLeptonXLiteThemeBundles.Styles.Global,
        bundle =>
    {
        bundle.AddFiles("/blazor-global-styles.css");
        //You can remove the following line if you don't use Blazor CSS isolation for
        components
        bundle.AddFiles("/MyProjectName.Blazor.styles.css");
    });
```

```

- Update `_Host.cshtml` file. *(located under **`Pages`** folder by default.)*

- Add following usings to Locate **`App`** and **`BlazorLeptonXLiteThemeBundles`** classes.
 

```
```csharp
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite
@using Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme.Bundling
```

```

- Then replace script & style bundles as following:

```
```diff
// Remove following line
```

```

```

- <abp-style-bundle name="@BlazorBasicThemeBundles.Styles.Global" />
// Add following line instead
+ <abp-style-bundle name="@BlazorLeptonXLiteThemeBundles.Styles.Global" />
```

```diff
// Remove following line
- <abp-script-bundle name="@BlazorBasicThemeBundles.Scripts.Global" />
// Add following line instead
+ <abp-script-bundle name="@BlazorLeptonXLiteThemeBundles.Scripts.Global" />
```

{{end}}

```

---

## ## Customization

### #### Layout

\* Create a razor page, like `MyMainLayout.razor`, in your blazor application as shown below:

```

```html
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite;
@using Volo.Abp.DependencyInjection

@inherits MainLayout
@attribute [ExposeServices(typeof(MainLayout))]
@attribute [Dependency(ReplaceServices = true)]

@Name
```

```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyMainLayout.razor.cs`, in your blazor application as shown below:

```

```csharp
[ExposeServices(typeof(MainLayout))]
[Dependency(ReplaceServices = true)]
namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    public partial class MyMainLayout
    {
        public string Name = "My Main Layout";
    }
}
```

```

> Don't forget to remove the repeated attributes from the razor page!

### ### Toolbars

LeptonX Lite includes separated toolbars for desktop & mobile. You can manage toolbars independently. Toolbar names can be accessible in the **\*\*LeptonXLiteToolbars\*\*** class.

```
- `LeptonXLiteToolbars.Main`
- `LeptonXLiteToolbars.MainMobile`  
  
```csharp
public async Task ConfigureToolbarAsync(IToolbarConfigurationContext context)
{
    if (context.Toolbar.Name == LeptonXLiteToolbars.Main)
    {
        context.Toolbar.Items.Add(new ToolbarItem(typeof(MyDesktopComponent)));
    }

    if (context.Toolbar.Name == LeptonXLiteToolbars.MainMobile)
    {
        context.Toolbar.Items.Add(new ToolbarItem(typeof(MyMobileComponent)));
    }

    return Task.CompletedTask;
}
```
{{if UI == "BlazorServer"}}

> _You can visit the \[Toolbars Documentation\] (https://docs.abp.io/en/abp/latest/UI/Blazor/Toolbars) for better understanding._  

{{end}}  
  
## Components
```

LeptonX Blazor is built on the basis of components. You can use the components in your application as you wish, or you can customize the components by overriding them. If you want to override a component please follow the steps.

### ### Branding Component

The **\*\*branding component\*\*** is a simple component that can be used to display your brand. It contains a **\*\*logo\*\*** and a **\*\*company name\*\***.

```

```

#### #### How to Override Branding Component

\* Create a razor page, like `MyBrandingComponent.razor`, in your blazor application as shown below:

```
```html
```

```

@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite;
@using Volo.Abp.DependencyInjection

@inherits Branding
@attribute [ExposeServices(typeof(Branding))]
@attribute [Dependency(ReplaceServices = true)]

@Name
```

```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyBrandingComponent.razor.cs`, in your blazor application as shown below:

```

```csharp
namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    public partial class MyBrandingComponent
    {
        public string Name = "My Branding Component";
    }
}
```

```

#### #### How to override the favicon

Startup templates contain `favicon.ico` files under the `wwwroot` folder of the Blazor application. You can change this file to override the current favicon.

#### #### Breadcrumb Component

On websites that have a lot of pages, **\*\*breadcrumb navigation\*\*** can greatly **\*\*enhance the way users find their way\*\*** around. In terms of **\*\*usability\*\***, breadcrumbs reduce the number of actions a website **\*\*visitor\*\*** needs to take in order to get to a **\*\*higher-level page\*\***, and they **\*\*improve\*\*** the **\*\*findability\*\*** of **\*\*website sections\*\*** and **\*\*pages\*\***.



#### ##### How to Override the BreadCrumb Component

\* Create a razor page, like `MyBreadcrumbsComponent.razor`, in your blazor application as shown below:

```

```html
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite;
@using Volo.Abp.DependencyInjection

@inherits Breadcrumbs
@attribute [ExposeServices(typeof(Breadcrumbs))]
@attribute [Dependency(ReplaceServices = true)]

@Name

```

```
```
```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyBreadcrumbsComponent.razor.cs`, in your blazor application as shown below:

```
*
```

```
```csharp
```

```
using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite;
```

```
using Volo.Abp.DependencyInjection;
```

```
namespace LeptonXLite.DemoApp.Blazor.MyComponents
```

```
{
```

```
    [ExposeServices(typeof(Breadcrumbs))]
```

```
    [Dependency(ReplaceServices = true)]
```

```
    public partial class MyBreadcrumbsComponent
```

```
    {
```

```
        public string Name = "My Breadcrumbs Component";
```

```
    }
```

```
}
```

```
```
```

#### ### Main Menu Component

Sidebar menus have been used as **\*\*a directory for Related Pages\*\*** for a **\*\*Service\*\*** offering, **\*\*Navigation\*\*** items for a **\*\*specific service\*\*** or topic and even just as **\*\*Links\*\*** the user may be interested in.

```

```

#### #### How to Override the Main Menu Component

\* Create a razor page, like `MyMainMenuComponent.razor`, in your blazor application as shown below:

```
```html
```

```
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite.Navigation;
```

```
@using Volo.Abp.DependencyInjection
```

```
@inherits MainMenu
```

```
@attribute [ExposeServices(typeof(MainMenu))]
```

```
@attribute [Dependency(ReplaceServices = true)]
```

```
@Name
```

```
```
```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyMainMenu.razor.cs`, in your blazor application as shown below:

```
```csharp
```

```
using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite.Navigation;
```

```
using Volo.Abp.DependencyInjection;
```

```

namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    [ExposeServices(typeof(MainMenu))]
    [Dependency(ReplaceServices = true)]
    public partial class MainMenu
    {
        public string Name = "My Main Menu Component";
    }
}
```

```

> The **main menu** renders the menu items **dynamically**. The **menu item** is a **razor component** named `MainMenu.razor.cs` in the same namespace with **main menu** and you can **override it** like the main menu.

#### ### Toolbar Items Component

Toolbar items are used to add **extra functionality to the toolbar**. The toolbar is a **horizontal bar** that **contains** a group of **toolbar items**.

#### #### How to Override the Toolbar Items Component

\* Create a razor page, like `MyToolbarItemsComponent.razor`, in your blazor application as shown below:

```

```html
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite;
@using Volo.Abp.DependencyInjection

@inherits ToolbarItemsComponent
@attribute [ExposeServices(typeof(ToolbarItemsComponent))]
@attribute [Dependency(ReplaceServices = true)]

@Name
```

```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyToolbarItemsComponent.razor.cs`, in your blazor application as shown below:

```

```csharp
using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite;
using Volo.Abp.DependencyInjection;

namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    [ExposeServices(typeof(ToolbarItemsComponent))]
    [Dependency(ReplaceServices = true)]
    public partial class MyToolbarItemsComponent
    {
        public string Name = "My Toolbar Items Component";
    }
}

```

```
}
```

### ### Language Switch Component

Think about a **multi-lingual** website and the first thing that could **hit your mind** is **the language switch component**. A **navigation bar** is a **great place** to **embed a language switch**. By embedding the language switch in the navigation bar of your website, you would **make it simpler** for users to **find it** and **easily** switch the **language** **<u>****without trying to locate it across the website.****</u>**

```

```

#### #### How to Override the Language Switch Component

\* Create a razor page, like `MyLanguageSwitchComponent.razor`, in your blazor application as shown below:

```
```html
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
@using Volo.Abp.DependencyInjection

@inherits LanguageSwitchComponent
@attribute [ExposeServices(typeof(LanguageSwitchComponent))]
@attribute [Dependency(ReplaceServices = true)]

@Name
````
```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyLanguageSwitchComponent.razor.cs`, in your blazor application as shown below:

```
```csharp
using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
using Volo.Abp.DependencyInjection;

namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    [ExposeServices(typeof(LanguageSwitchComponent))]
    [Dependency(ReplaceServices = true)]
    public partial class MyLanguageSwitchComponent
    {
        public string Name = "My Language Switch Component";
    }
}
````
```

### ### Mobile Language Switch Component

The **mobile language switch component** is used to switch the language of the website **on mobile devices**. The mobile language switch component is a **dropdown menu** that **contains all the languages** of the website.

```

```

#### #### How to Override the Mobile Language Switch Component

\* Create a razor page, like `MyMobilLanguageSwitchComponent.razor`, in your blazor application as shown below:

```
```html
@using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
@using Volo.Abp.DependencyInjection

@inherits MobilLanguageSwitchComponent
@attribute [ExposeServices(typeof(MobilLanguageSwitchComponent))]
@attribute [Dependency(ReplaceServices = true)]

@Name
```

```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyMobilLanguageSwitchComponent.razor.cs`, in your blazor application as shown below:

```
```csharp
using Volo.Abp.AspNetCore.Components.Web.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
using Volo.Abp.DependencyInjection;

namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    [ExposeServices(typeof(MobilLanguageSwitchComponent))]
    [Dependency(ReplaceServices = true)]
    public partial class MyMobilLanguageSwitchComponent
    {
        public string Name = "My Mobile Language Switch Component";
    }
}
```

```

#### ### User Menu Component

The **User Menu** is the **menu** that **drops down** when you **click your name** or **profile picture** in the **upper right corner** of your page (**in the toolbar**). It drops down options such as **Settings**, **Logout**, etc.

```

```

#### #### How to Override the User Menu Component

\* Create a razor page, like `MyUserMenuComponent.razor`, in your blazor application as shown below:

```
```html
@using Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
@using Volo.Abp.DependencyInjection

@inherits MobilLanguageSwitchComponent
@attribute [ExposeServices(typeof(MobilLanguageSwitchComponent))]
@attribute [Dependency(ReplaceServices = true)]

@Name
```

```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyUserMenuComponent.razor.cs`, in your blazor application as shown below:

```
```csharp
using Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
using Volo.Abp.DependencyInjection;

namespace LeptonXLite.DemoApp.Blazor.MyComponents
{
    [ExposeServices(typeof(UserMenuComponent))]
    [Dependency(ReplaceServices = true)]
    public partial class MyUserMenuComponent
    {
        public string Name = "My User Menu Component";
    }
}
```

```

#### #### Mobile User Menu Component

The **mobile user menu component** is used to display the **user menu on mobile devices**. The mobile user menu component is a **dropdown menu** that contains all the **options** of the **user menu**.

```

```

#### #### How to override the Mobile User Menu Component

\* Create a razor page, like `MyMobileUserMenuComponent.razor`, in your blazor application as shown below:

```
```html
@using Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;
@using Volo.Abp.DependencyInjection

@inherits MobilUserMenuComponent
@attribute [ExposeServices(typeof(MobilUserMenuComponent))]
```

```
@attribute [Dependency(ReplaceServices = true)]
```

```
@Name  
````
```

\* If you prefer to use a code-behind file for the C# code of your component, create a razor component, like `MyMobileUserMenuComponent.razor.cs`, in your blazor application as shown below:

```
```csharp  
using Volo.Abp.AspNetCore.Components.Server.LeptonXLiteTheme.Themes.LeptonXLite.Toolbar;  
using Volo.Abp.DependencyInjection;  
  
namespace LeptonXLite.DemoApp.Blazor.MyComponents  
{  
    [ExposeServices(typeof(MobileUserMenuComponent))]  
    [Dependency(ReplaceServices = true)]  
    public partial class MyMobileUserMenuComponent  
    {  
        public string Name = "My Mobile User Menu Component";  
    }  
}
```

#### 10.2.4.4 Branding

```
# Blazor UI: Branding
```

```
## IBrandingProvider
```

`IBrandingProvider` is a simple interface that is used to show the application name and logo on the layout.

The screenshot below shows \*MyProject\* as the application name:

```
![branding-nobrand](../../images/branding-nobrand.png)
```

You can implement the `IBrandingProvider` interface or inherit from the `DefaultBrandingProvider` to set the application name:

```
```csharp  
using Volo.Abp.DependencyInjection;  
using Volo.Abp.Ui.Branding;  
  
namespace MyCompanyName.MyProjectName.Blazor  
{  
    [Dependency(ReplaceServices = true)]  
    public class MyProjectNameBrandingProvider : DefaultBrandingProvider  
    {  
        public override string AppName => "Book Store";
```

```
    }  
}  
```
```

The result will be like shown below:

```
![branding-appname](../../images/branding-appname.png)
```

`IBrandingProvider` has the following properties:

- \* `AppName`: The application name.
- \* `LogoUrl`: A URL to show the application logo.
- \* `LogoReverseUrl`: A URL to show the application logo on a reverse color theme (dark, for example).

> **\*\*Tip\*\*:** `IBrandingProvider` is used in every page refresh. For a multi-tenant application, you can return a tenant specific application name to customize it per tenant.

#### 10.2.4.5 Page Header

##### # Blazor UI: Page Header

You can use the `PageHeader` component to set the page title, the breadcrumb items and the toolbar items for a page. Before using the `PageHeader` component, you need to add a using statement for the `Volo.Abp.AspNetCore.Components.Web.Theming.Layout` namespace.

Once you add the `PageHeader` component to your page, you can control the related values using the parameters.

##### ## Page Title

You can use the `Title` parameter to control the page header.

```
```csharp  
<PageHeader Title="Book List">  
</PageHeader>  
```
```

##### ## Breadcrumb

> **\*\*The [Basic Theme](Basic-Theme.md) currently doesn't implement the breadcrumbs.\*\***  
>  
> The [LeptonX Lite Theme](../../Themes/LeptonXLite/Blazor.md) supports breadcrumbs.

Breadcrumbs can be added using the `BreadcrumbItems` property.

##### **\*\*Example: Add Language Management to the breadcrumb items.\*\***

Create a collection of `Volo.Abp.BlazoriseUI.BreadcrumbItem` objects and set the collection to the `BreadcrumbItems` parameter.

```

```csharp
public partial class Index
{
    protected List<BreadcrumbItem> BreadcrumbItems { get; } = new();

    protected override void OnInitialized()
    {
        BreadcrumbItems.Add(new BreadcrumbItem("Language Management"));
    }
}
```

```

Navigate back to the razor page.

```

```csharp
<PageHeader BreadcrumbItems="@BreadcrumbItems" />
```

```

The theme then renders the breadcrumb. An example render result can be:

```

![breadcrumbs-example](../../images/breadcrumbs-example.png)

* The Home icon is rendered by default. Set `BreadcrumbShowHome` to `false` to hide it.
* Breadcrumb items will be activated based on current navigation. Set `BreadcrumbShowCurrent` to `false` to disable it.

```

You can add as many items as you need. `BreadcrumbItem` constructor gets three parameters:

- \* `text`: The text to show for the breadcrumb item.
- \* `url` (optional): A URL to navigate to, if the user clicks to the breadcrumb item.
- \* `icon` (optional): An icon class (like `fas fa-user-tie` for Font-Awesome) to show with the `text`.

## ## Page Toolbar

Page toolbar can be set using the `Toolbar` property.

### **\*\*Example: Add a "New Item" toolbar item to the page toolbar.\*\***

Create a `PageToolbar` object and define toolbar items using the `AddButton` extension method.

```

```csharp
public partial class Index
{
    protected PageToolbar Toolbar { get; } = new();

    protected override void OnInitialized()
    {
        Toolbar.AddButton("New Item", () =>
        {

```

```
        //Write your click action here
        return Task.CompletedTask;
    }, icon:IconName.Add);
}
```

```

Navigate back to the razor page and set the `Toolbar` parameter.

```
```csharp
<PageHeader Toolbar="@Toolbar" />
```

```

An example render result can be:

```
![breadcrumbs-example](../../images/page-header-toolbar-blazor.png)
```

---

#### ## Options

Rendering can be enabled or disabled for each section of PageHeader via using `PageHeaderOptions`.

```
```csharp
Configure<PageHeaderOptions>(options =>
{
    options.RenderPageTitle = false;
    options.RenderBreadcrumbs = false;
    options.RenderToolbar = false;
});
```

```

*\*All values are **\*\*true\*\*** by default. If the PageHeaderOptions isn't configured, each section will be rendered.\**

#### 10.2.4.6 Page Layout

##### # Page Layout

PageLayout is used to define page-specific elements across application.

##### ## Title

Title is used to render page title in the PageHeader.

```
```csharp
@inject PageLayout PageLayout

 @{
    PageLayout.Title = "My Page Title";
}
```

```

## ## MenuItemName

Indicates current selected menu item name. Menu item name should match a unique menu item name defined using the [Navigation / Menu system](../Blazor/Navigation-Menu.md). In this case, it is expected from the theme to make the menu item "active" in the main menu.

```
```csharp
@inject PageLayout PageLayout

@code {
    protected override async Task OnInitializedAsync()
    {
        PageLayout.MenuItemName = "MyProjectName.Products";
    }
}
```

```

Menu item name can be set on runtime too.

```
```html
@inject PageLayout PageLayout

<Button Clicked="SetCategoriesMenuAsSelected">Change Menu</Button>

@code{
    protected void SetCategoriesMenuAsSelected()
    {
        PageLayout.MenuItemName = "MyProjectName.Categories";
    }
}
```

```

![leptonx selected menu item](../../images/leptonx-selected-menu-item-example.gif)

➤ Be aware, The [Basic Theme](../Blazor/Basic-Theme.md) currently doesn't support the selected menu item since it is not applicable to the top menu.

## ## BreadCrumbs

BreadCrumbItems are used to render breadcrumbs in the PageHeader.

```
```csharp
@{
    PageLayout.BreadcrumbItems.Add(new BlazoriseUI.BreadcrumbItem("My Page", "/my-page"));
}
```

```

## ## Toolbar

ToolbarItems are used to render action toolbar items in the PageHeader.

Check out [Page Toolbar] (<https://docs.abp.io/en/abp/latest/UI/Blazor/Page-Header#page-toolbar>)

```
```csharp
@inject PageLayout PageLayout
#{@
    PageLayout.ToolbarItems.Add(new
PageToolbars.PageToolbarItem(typeof(MyButtonComponent)));
}
```

```

#### 10.2.4.7 Toolbars

##### # Blazor UI: Toolbars

The Toolbar system is used to define **\*\*toolbars\*\*** on the user interface. Modules (or your application) can add **\*\*items\*\*** to a toolbar, then the [theme] (Theming.md) renders the toolbar on the **\*\*layout\*\***.

There is only one **\*\*standard toolbar\*\*** named "Main" (defined as a constant: `StandardToolbars.Main`). The [Basic Theme] (Basic-Theme) renders the main toolbar as shown below:

```
![bookstore-toolbar-highlighted](../../images/bookstore-toolbar-highlighted.png)
```

In the screenshot above, there are two items added to the main toolbar: Language switch component & user menu. You can add your own items here.

Also, [LeptonX Lite Theme] (../../Themes/LeptonXLite/Blazor.md) has 2 different toolbars for desktop and mobile views which defined as constants: `LeptonXLiteToolbars.Main`, `LeptonXLiteToolbars.MainMobile`.

```
| LeptonXLiteToolbars.Main | LeptonXLiteToolbars.MainMobile |
| :---: | :---: |
| !*[leptonx](../../images/leptonxlite-toolbar-main-example.png)
| !*[leptonx](../../images/leptonxlite-toolbar-mainmobile-example.png) |
```

##### ## Example: Add a Notification Icon

In this example, we will add a **\*\*notification (bell) icon\*\*** to the left of the language switch item. A item in the toolbar should be a **\*\*Razor Component\*\***. So, first, create a new razor component in your project (the location of the component doesn't matter):

```
![bookstore-notification-view-component](../../images/blazor-notification-bell-component.png)
```

The content of the `Notification.razor` is shown below:

```
```html
@inherits Volo.Abp.AspNetCore.Components.AbpComponentBase
<div style="color: white; margin: 8px;">
```

```

        <i class="far fa-bell" @onclick="ShowNotifications"></i>
    </div>
@code {
    private async Task ShowNotifications()
    {
        await Message.Info("TODO: Show notifications");
    }
}
````
```

This sample simply shows a message. In real life, you probably want to call an HTTP API to get notifications and show on the UI.

Now, we can create a class implementing the `IToolbarContributor` interface:

```

```csharp
using Volo.Abp.AspNetCore.Components.Web.Theming.Toolbars;

// ...

public class MyToolbarContributor : IToolbarContributor
{
    public Task ConfigureToolbarAsync(IToolbarConfigurationContext context)
    {
        if (context.Toolbar.Name == StandardToolbars.Main)
        {
            context.Toolbar.Items.Insert(0, new ToolbarItem(typeof(Notification)));
        }

        return Task.CompletedTask;
    }
}
````
```

This class adds the `NotificationViewComponent` as the first item in the `Main` toolbar.

Finally, you need to add this contributor to the `AbpToolbarOptions`, in the `ConfigureServices` of your [module](`./Module-Development-Basics.md`):

```

```csharp
using Volo.Abp.AspNetCore.Components.Web.Theming.Toolbars;
````

```csharp
Configure<AbpToolbarOptions>(options =>
{
    options CONTRIBUTORS.Add(new MyToolbarContributor());
});
````
```

That's all, you will see the notification icon on the toolbar when you run the application:

```
![bookstore-notification-icon-on-toolbar](../../images/bookstore-notification-icon-on-toolbar.png)
```

## ## IToolbarManager

`IToolbarManager` is used to render the toolbar. It returns the toolbar items by a toolbar name. This is generally used by the [themes] (Theming.md) to render the toolbar on the layout.

### 10.2.5 Security

#### 10.2.5.1 Authentication

##### # Blazor UI: Authentication

```
```json
//[doc-params]
{
    "UI": ["Blazor", "BlazorServer"]
}
```
```

The [application startup template] (../../Startup-Templates/Application.md) is properly configured to use OpenId Connect to authenticate the user;

```
 {{if UI == "BlazorServer"}}
```

The Blazor Server application UI is actually a hybrid application that is combined with the MVC UI, and uses the login page provided by the MVC UI. When users enter a page that requires login, they are redirected to the `/Account/Login` page. Once they complete the login process, they are returned back to the application's UI. The login page also contains features like registration, password recovery, etc.

```
 {{end}}
```

```
 {{if UI == "Blazor"}}
```

- \* When the Blazor application needs to authenticate, it is redirected to the server side.
- \* Users can enter username & password to login if they already have an account. If not, they can use the register form to create a new user. They can also use forgot password and other features. The server side uses OpenIddict to handle the authentication.
- \* Finally, they are redirected back to the Blazor application to complete the login process.

This is a typical and recommended approach to implement authentication in Single-Page Applications. The client side configuration is done in the startup template, so you can change it.

See the [Blazor Security document] (<https://docs.microsoft.com/en-us/aspnet/core/blazor/security>) to understand and customize the authentication process.

```
 {{end}}
```

#### 10.2.5.2 Authorization

##### # Blazor UI: Authorization

Blazor applications can use the same authorization system and permissions defined in the server side.

> This document is only for authorizing on the Blazor UI. See the [Server Side Authorization](../../Authorization.md) to learn how to define permissions and control the authorization system.

##### ## Basic Usage

> ABP Framework is **\*\*100% compatible\*\*** with the Authorization infrastructure provided by the Blazor. See the [Blazor Security Document](https://docs.microsoft.com/en-us/aspnet/core/blazor/security/) to learn all authorization options. This section **\*\*only shows some common scenarios\*\***.

##### ### Authorize Attribute

`[Authorize]` attribute can be used to show a page only to the authenticated users.

```
```csharp
@page "/"
@attribute [Authorize]
```

You can only see this if you're signed in.

```
```
```

The `[Authorize]` attribute also supports role-based or policy-based authorization. For example, you can check permissions defined in the server side:

```
```csharp
@page "/"
@attribute [Authorize("MyPermission")]
```

You can only see this if you have the necessary permission.

```
```
```

##### ### AuthorizeView

`AuthorizeView` component can be used in a page/component to conditionally render a part of the content:

```
```html
<AuthorizeView Policy="MyPermission">
    <p>You can only see this if you satisfy the "MyPermission" policy.</p>
</AuthorizeView>
```
```

##### ### IAuthorizationService

`IAuthorizationService` can be injected and used to programmatically check permissions:

```
```csharp
public partial class Index
{
    protected override async Task OnInitializedAsync()
    {
        if (await AuthorizationService.IsGrantedAsync("MyPermission"))
        {
            //...
        }
    }
}```

```

If your component directly or indirectly inherits from the `AbpComponentBase`, `AuthorizationService` becomes pre-injected and ready to use. If not, you can always [inject](../../Dependency-Injection.md) the `IAuthorizationService` yourself.

`IAuthorizationService` can also be used in the view side where `AuthorizeView` component is not enough.

There are some useful extension methods for the `IAuthorizationService`:

- \* `IsGrantedAsync` simply returns `true` or `false` for the given policy/permission.
- \* `CheckAsync` checks and throws `AbpAuthorizationException` if given policy/permission hasn't granted. You don't have to handle these kind of exceptions since ABP Framework automatically [handles errors](Error-Handling.md).
- \* `AuthorizeAsync` returns `AuthorizationResult` as the standard way provided by the ASP.NET Core authorization system.

> See the [Blazor Security Document](<https://docs.microsoft.com/en-us/aspnet/core/blazor/security/>) to learn all authorization options

## ## See Also

- \* [Authorization](../../Authorization.md) (server side)
- \* [Blazor Security](<https://docs.microsoft.com/en-us/aspnet/core/blazor/security/>) (Microsoft documentation)
- \* [ICurrentUserService](ICurrentUserService.md) (CurrentUser.md)

## 10.2.6 Services

### 10.2.6.1 Current User

# Blazor UI: Current User

`ICurrentUser` service is used to obtain information about the currently authenticated user. Inject the `ICurrentUser` into any component/page and use its properties and methods.

**\*\*Example: Show username & email on a page\*\***

```
```csharp
@page "/"
@using Volo.Abp.Users
@inject ICurrentUser CurrentUser
@if (CurrentUser.IsAuthenticated)
{
    <p>Welcome @CurrentUser.UserName</p>
}
```
```

> If you (directly or indirectly) derived your component from the `AbpComponentBase`, you can directly use the base `CurrentUser` property.

`ICurrentUser` provides `Id`, `Name`, `SurName`, `Email`, `Roles` and some other properties.

> See the [Server Side Current User](../../../../CurrentUser) service for more information.

#### 10.2.6.2 Current Tenant

##### # Blazor UI: Current Tenant

`ICurrentTenant` service can be used to get information about the current tenant in a [multi-tenant](../../../../Multi-Tenancy.md) application. `ICurrentTenant` defines the following properties;

- \* `Id` (`Guid`): Id of the current tenant. Can be `null` if the current user is a host user or the tenant could not be determined.
- \* `Name` (`string`): Name of the current tenant. Can be `null` if the current user is a host user or the tenant could not be determined.
- \* `IsAvailable` (`bool`): Returns `true` if the `Id` is not `null`.

**\*\*Example: Show the current tenant name on a page\*\***

```
```csharp
@page "/"
@using Volo.Abp.MultiTenancy
@inject ICurrentTenant CurrentTenant
@if (CurrentTenant.IsAvailable)
{
    <p>Current tenant name: @CurrentTenant.Name</p>
}
```
```

## See Also

\* [Multi-Tenancy](../../Multi-Tenancy.md)

#### 10.2.6.3 Notification

# Blazor UI: Notification Service

`IUiNotificationService` is used to show toast style notifications on the user interface.

## Quick Example

Simply [inject](../../Dependency-Injection.md) `IUiNotificationService` to your page or component and call the `Success` method to show a success message.

```
```csharp
namespace MyProject.Blazor.Pages
{
    public partial class Index
    {
        private readonly IUiNotificationService _uiNotificationService;

        public Index(IUiNotificationService uiNotificationService)
        {
            _uiNotificationService = uiNotificationService;
        }

        public async Task DeleteAsync()
        {
            await _uiNotificationService.Success(
                "The product 'Acme Atom Re-Arranger' has been successfully deleted.");
        }
    }
}
````
```

![blazor-notification-sucess](../../images/blazor-notification-success.png)

If you inherit your page or component from the `AbpComponentBase` class, you can use the the `Notify` property to access the `IUiNotificationService` as a pre-injected property.

```
```csharp
namespace MyProject.Blazor.Pages
{
    public partial class Index : AbpComponentBase
    {
        public async Task DeleteAsync()
        {
            await Notify.Success(

```

```

        );
    }
}
```

```

> You typically use `@inherits AbpComponentBase` in the `\*.razor` file to inherit from the `AbpComponentBase`, instead of inheriting in the code behind file.

## ## Notification Types

There are four types of pre-defined notifications;

- \* `Info(...)`
- \* `Success(...)`
- \* `Warn(...)`
- \* `Error(...)`

All of the methods above gets the following parameters;

- \* `message` : The message (`string`) to be shown.
- \* `title` : An optional (`string`) title.
- \* `options` : An optional (`Action`) to configure notification options.

## ## Configuration

### ### Per Notification

It is easy to change default notification options if you like to customize it per notification. Provide an action to the `options` parameter as shown below:

```

```csharp
await UiNotificationService.Success(
    "The product 'Acme Atom Re-Arranger' has been successfully deleted.",
    options: (options) =>
    {
        options.OkButtonText =
            LocalizableString.Create<MyProjectNameResource>("CustomOK");
    });
```

```

### ### Available Options

Here, the list of all available options;

- \* `OkButtonText` : Custom text for the OK button.
- \* `OkButtonIcon` : Custom icon for the OK button

### ### Global Configuration

You can also configure global notification options to control the it in a single point. Configure the `UiNotificationOptions` [options class](../../Options.md) in the `ConfigureServices` of your [module](../../Module-Development-Basics.md):

```
```csharp
Configure<UiNotificationOptions>(options =>
{
    options.OkButtonText = LocalizableString.Create<MyProjectNameResource>("CustomOK");
});
```
```

The same options are available here.

> \*Per notification\* configuration overrides the default values.

#### 10.2.6.4 Message

##### # Blazor UI: Message Service

UI message service is used to show nice-looking messages to the user as a blocking dialog.

##### ## Quick Example

Simply [inject](../../Dependency-Injection.md) `IUiMessageService` to your page or component and call the `Success` method to show a success message.

```
```csharp
namespace MyProject.Blazor.Pages
{
    public partial class Index
    {
        private readonly IUiMessageService _uiMessageService;

        public Index(IUiMessageService uiMessageService)
        {
            _uiMessageService = uiMessageService;
        }

        public async Task SaveAsync()
        {
            await _uiMessageService.Success(
                "Your changes have been successfully saved!",
                "Congratulations");
        }
    }
}
```

It will show a dialog on the UI:

![blazor-message-success](../../images/blazor-message-success.png)

If you inherit your page or component from the `AbpComponentBase` class, you can use the `Message` property to access the `IUiMessageService` as a pre-injected property.

```
```csharp
namespace MyProject.Blazor.Pages
{
    public partial class Index : AbpComponentBase
    {
        public async Task SaveAsync()
        {
            await Message.Success(
                "Your changes have been successfully saved!",
                "Congratulations");
        }
    }
}
```

```

> You typically use `@inherits AbpComponentBase` in the `\*.razor` file to inherit from the `AbpComponentBase`, instead of inheriting in the code behind file.

## ## Informative Messages

There are four types of informative message functions:

- \* `Info(...)`
- \* `Success(...)`
- \* `Warn(...)`
- \* `Error(...)`

All of these methods get three parameters:

- \* `message`: The message (`string`) to be shown.
- \* `title`: An optional (`string`) title.
- \* `options`: An optional (`Action`) to configure UI message options.

### **\*\*Example: Show an error message\*\***

```
```csharp
_iuiMessageService.Error('Your credit card number is not valid!');
```

```

**![blazor-message-success](../../images/blazor-message-error.png)**

## ## Confirmation Message

`IUiMessageService.Confirm(...)` method can be used to get a confirmation from the user.

### **\*\*Example\*\***

Use the following code to get a confirmation result from the user:

```

```csharp
public async Task DeleteAsync()
{
    var confirmed = await _uiMessageService.Confirm("Are you sure to delete the 'admin' role?");
    if(confirmed)
    {
        //Delete the 'admin' role here.
    }
}
```

```

The resulting UI will be like shown below:

`![blazor-message-confirm](../../images/blazor-message-confirm.png)`

If the user has clicked the `Yes` button, the `Confirm` method's return value will be `true`.

## ## Configuration

It is easy to change default message options if you like to it per message. Provide an `action` to the `options` parameter as shown below.

```

```csharp
await _uiMessageService.Success(
    "Your changes have been successfully saved!",
    "Congratulations",
    (options) =>
    {
        options.MessageIcon = "msg-icon-new";
        options.CenterMessage = false;
    });
```

```

List of the options that you can change by providing the `action` parameter.

- \* `CenterMessage` : (Default: true) If true, the message dialogue will be centered on the screen.
- \* `ShowMessageIcon` : (Default: true) If true, the message dialogue will show the large icon for the current message type.
- \* `MessageIcon` : Overrides the build-in message icon.
- \* `OkButtonText` : Custom text for the OK button.
- \* `OkButtonIcon` : Custom icon for the OK button.
- \* `ConfirmButtonText` : Custom text for the Confirmation button.
- \* `ConfirmButtonIcon` : Custom icon for the Confirmation button.
- \* `CancelButtonText` : Custom text for the Cancel button.
- \* `CancelButtonIcon` : Custom icon for the Cancel button.

> "Confirm", "Cancel" and "Yes" texts are automatically localized based on the current language.

#### 10.2.6.5 Page Alerts

```
# Blazor UI: Page Alerts
```

It is common to show error, warning or information alerts to inform the user. An example **\*Service Interruption\*** alert is shown below:

```
![blazor-page-alert-example](../../images/blazor-page-alert-example.png)
```

```
## Quick Example
```

Simply `[inject]`(../../Dependency-Injection.md) `IAlertManager` to your page or component and call the `Alerts.Warning` method to show a success message.

```
```csharp
namespace MyProject.Blazor.Pages
{
    public partial class Index
    {
        private readonly IAlertManager _alertManager;

        public Index(IAlertManager alertManager)
        {
            this._alertManager = alertManager;
        }

        protected override void OnInitialized()
        {
            _alertManager.Alerts.Warning(
                "We will have a service interruption between 02:00 AM and 04:00 AM at
October 23, 2023!",
                "Service Interruption");
            base.OnInitialized();
        }
    }
}```
```

If you inherit your page or component from the `AbpComponentBase` class, you can use the `Alerts` property to add alerts.

```
```csharp
namespace MyProject.Blazor.Pages
{
    public partial class Index : AbpComponentBase
    {
        protected override void OnInitialized()
        {
            Alerts.Warning(
                "We will have a service interruption between 02:00 AM and 04:00 AM at
October 23, 2023!",
```

```

        "Service Interruption");
        base.OnInitialized();
    }
}
```

```

> You typically use `@inherits AbpComponentBase` in the `\*.razor` file to inherit from the `AbpComponentBase`, instead of inheriting in the code behind file.

#### #### Alert Types

`Warning` is used to show a warning alert. Other common methods are `Info`, `Danger` and `Success`.

Beside the standard methods, you can use the `Alerts.Add` method by passing an `AlertType` enum with one of these values: `Default`, `Primary`, `Secondary`, `Success`, `Danger`, `Warning`, `Info`, `Light`, `Dark`.

#### #### Dismissible

All alert methods gets an optional `dismissible` parameter. Default value is `true` which makes the alert box dismissible. Set it to `false` to create a sticky alert box.

### 10.2.6.6 Page Progress

#### # Blazor UI: Page Progress

Page Progress is used to show a progress bar indicator on top of the page and to show to the user that currently a long running process is in the work.

By default you don't need to do anything to show the progress indicator, as all the work is done automatically by the ABP Framework internals. This means that all calls to the ABP backend (through your HTTP API) will activate page progress and show the loading indicator.

This doesn't mean that you don't have the control over it. On the contrary, if you want to show progress for your own processes, it is really easy to do. All you have to do is to use inject and use the `IUiPageProgressService`.

#### ## Example

First, inject the `IUiPageProgressService` into your page/component.

```

```cs
@inject IUiPageProgressService pageProgressService
```

```

Next, invoke the `Go` method in `IUiPageProgressService`. It's that simple:

```

```cs
Task OnClick()
{
    return pageProgressService.Go(null);
}
```

```

The previous example will show the progress with a default settings. If, for example you want to change the progress color you can override it by setting the options through the `Go` method.

```

```cs
Task OnClick()
{
    return pageProgressService.Go(null, options =>
    {
        options.Type = UiPageProgressType.Warning;
    });
}
```

```

#### ## Breakdown

The first parameter of the `Go` needs a little explanation. In the previous example we have set it to `null` which means, once called it will show an *\_indeterminate\_* indicator and will cycle the loading animation indefinitely, until we hide the progress. You also have the option of defining the actual percentage of the progress and the code is the same, just instead of sending it the `null` you will send it a number between `0` and `100`.

```

```cs
pageProgressService.Go(25)
```

```

#### ### Valid values

1. `null` – show *\_indeterminate\_* indicator
2. `>= 0` and `<= 100` – show the regular *\_percentage\_* progress

#### ### Hiding progress

To hide the progress just set the actual values to something other then the *\_Valid value\_*.

```

```cs
pageProgressService.Go(-1)
```

```

### 10.2.7 Other Components

#### 10.2.7.1 SubmitButton

# Blazor UI: SubmitButton component

`SubmitButton` is a simple wrapper around `Button` component. It is used to be placed inside of page Form or Modal dialogs where it can response to user actions and to be activated as a default button by pressing an ENTER key. Once clicked it will go into the `disabled` state and also it will show a small loading indicator until clicked event is finished.

#### ## Quick Example

```
```html
<SubmitButton Clicked="@YourSaveOperation" />
````
```

Notice that we didn't specify any text, like `Save Changes`. This is because `SubmitButton` will by default pull text from the localization. If you want to change that you either specify a localization key or you can add custom content.

#### ### With localization key

```
```html
<SubmitButton Clicked="@YourSaveOperation" SaveResourceKey="YourSaveName" />
````
```

#### ### With custom content

```
```html
<SubmitButton Clicked="@YourSaveOperation">
    @L["Save"]
</SubmitButton>
````
```

### 10.2.8 Settings

#### # Blazor UI: Settings

Blazor applications can reuse the same `ISettingProvider` service that is explained in the [settings document](../../Settings.md).

#### ## ISettingProvider

`ISettingProvider` is used to get the value of a setting or get the values of all the settings.

#### **\*\*Example usages in a simple service\*\***

```
```csharp
public class MyService : ITransientDependency
{
    private readonly ISettingProvider _settingProvider;

    //Inject ISettingProvider in the constructor
```

```

public MyService(ISettingProvider settingProvider)
{
    _settingProvider = settingProvider;
}

public async Task FooAsync()
{
    //Get a value as string.
    string setting1 = await _settingProvider.GetOrNullAsync("MySettingName");

    //Get a bool value and fallback to the default value (false) if not set.
    bool setting2 = await _settingProvider.GetAsync<bool>("MyBoolSettingName");

    //Get a bool value and fallback to the provided default value (true) if not set.
    bool setting3 = await _settingProvider.GetAsync<bool>(
        "MyBoolSettingName", defaultValue: true);

    //Get a bool value with the IsTrueAsync shortcut extension method
    bool setting4 = await _settingProvider.IsTrueAsync("MyBoolSettingName");

    //Get an int value or the default value (0) if not set
    int setting5 = (await _settingProvider.GetAsync<int>("MyIntegerSettingName"));

    //Get an int value or null if not provided
    int? setting6 = (await _settingProvider
        .GetOrNullAsync("MyIntegerSettingName"))?.To<int>();
}
```
```

```

**\*\*Example usage in a Razor Component\*\***

```

```csharp
@page "/"
@using Volo.Abp.Settings
@inject ISettingProvider SettingProvider
@code {
    protected override async Task OnInitializedAsync()
    {
        bool settingValue = await SettingProvider.GetAsync<bool>("MyBoolSettingName");
    }
}
```

```

## See Also

- \* [Settings](../../Settings.md)

## 10.2.9 Error Handling

# Blazor UI: Error Handling

```

```json
//[doc-params]
{
    "UI": ["Blazor", "BlazorServer"]
}
```

```

Blazor, by default, shows a yellow line at the bottom of the page if any unhandled exception occurs. However, this is not useful in a real application.

ABP provides an automatic error handling system for the Blazor UI.

- \* Handles all unhandled exceptions and shows nice and useful messages to the user.
- \* It distinguishes different kind of exceptions. Hides internal/technical error details from the user (shows a generic error message in these cases).
- \* It is well integrated to the [server side exception handling](../../Exception-Handling.md) system.

#### ## Basic Usage

There are different type of `Exception` classes handled differently by the ABP Framework.

#### ### UserFriendlyException

`UserFriendlyException` is a special type of exception. You can directly show a error message dialog to the user by throwing such an exception.

```
{{if UI == "BlazorServer"}}
```

> For Blazor Server, exceptions must be handled manually. Otherwise it crashes the whole application. Auto Exception Handling is not possible with Blazor Server right now, please follow [this issue](https://github.com/abpframework/abp/issues/8195) to see progress. In meantime, you can use try-catch blocks and call the `HandleErrorAsync` method of ABP to handle errors manually, which shows an error dialog for you.

#### **\*\*Example\*\***

```

```csharp
@page "/"
@using Volo.Abp

<Button Clicked="TestException">Throw test exception</Button>

@code
{
    private async Task TestException()
    {
        try
        {
            throw new UserFriendlyException("A user friendly error message!");
        }
    }
}
```

```

```

        catch(UserFriendlyException ex)
        {
            await HandleErrorAsync(ex);
        }
    }
```
{{end}}

```

  {{if UI == "Blazor"}}

#### **\*\*Example\*\***

```

~~~~csharp
@page "/"
@using Volo.Abp

<Button Clicked="TestException">Throw test exception</Button>

@code
{
 private void TestException()
 {
 throw new UserFriendlyException("A user friendly error message!");
 }
```
{{end}}

```

ABP automatically handle the exception and show an error message to the user:

![blazor-user-friendly-exception](../../images/blazor-user-friendly-exception.png)

- You can derive from `UserFriendlyException` or directly implement `IUserFriendlyException` interface to create your own `Exception` class if you need.
- You can use the [localization system](Localization.md) to show localized error messages.

BusinessException and Other Exception Types

See the [exception handling document](../../Exception-Handling.md) to understand different kind of Exception class and interfaces and other capabilities of the Exception Handling system.

Generic Errors

If the thrown `Exception` is not a special type, it is considered as generic error and a generic error message is shown to the user:

![blazor-generic-exception-message](../../images/blazor-generic-exception-message.png)

› All error details (including stack trace) are still written in the browser's console.

Server Side Errors

Errors (like Validation, Authorization and User Friendly Errors) sent by the server are processed as you expect and properly shown to the user. So, error handling system works end to end without need to manually handle exceptions or manually transfer server-to-client error messages.

See Also

- * [Exception Handling System](../../Exception-Handling.md)

10.2.10 Customization / Overriding Components

Blazor UI: Customization / Overriding Components

```
```json
//[doc-params]
{
 "UI": ["Blazor", "BlazorServer"]
}
```
```

This document explains how to override the user interface of a depended [application module](../../Modules/Index.md) or [theme](Theming.md) for Blazor applications.

Overriding a Razor Component

The ABP Framework, pre-built themes and modules define some ****re=usable razor components and pages****. These pages and components can be replaced by your application or module.

› Since pages are just the razor components, the same principle is valid for pages too.

Example: Replacing the Branding Area

The screenshot below was taken from the [Basic Theme](Basic-Theme.md) comes with the application startup template.

![bookstore-brand-area-highlighted](../../images/bookstore-brand-area-highlighted.png)

The [Basic Theme](Basic-Theme.md) defines some razor components for the layout. For example, the highlighted area with the red rectangle above is called **Branding** component. You probably want to customize this component by adding your ****own application logo****. Let's see how to do it.

First, create your logo and place under a folder in your web application. We used `~wwwroot/bookstore-logo.png` path:

![bookstore-logo-blazor](../../images/bookstore-logo-blazor.png)

The next step is to create a razor component, like `MyBranding.razor`, in your application:

```
![bookstore-logo-blazor](../../images/bookstore-branding-blazor.png)
```

The content of the `MyBranding.razor` is shown below:

```
```html
@using Volo.Abp.DependencyInjection
{{if UI == "BlazorServer"}}
@using Volo.Abp.AspNetCore.Components.Server.BasicTheme.Themes.Basic
{{end}}
{{if UI == "Blazor"}}
@using Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme.Themes.Basic
{{end}}

@inherits Branding
@attribute [ExposeServices(typeof(Branding))]
@attribute [Dependency(ReplaceServices = true)]


````
```

Let's explain the code:

- * `@inherits Branding` line inherits the Branding component defined by the [Basic Theme](Basic-Theme.md) (in the {{if UI == "BlazorServer"}}`Volo.Abp.AspNetCore.Components.Server.BasicTheme.Themes.Basic` {{end}} {{if UI == "Blazor"}}`Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme.Themes.Basic` {{end}} namespace).
- * `@attribute [ExposeServices(typeof(Branding))]"` registers this service (component) to [dependency injection](../../Dependency-Injection.md) for the `Branding` service (component).
- * `@attribute [Dependency(ReplaceServices = true)]` replaces the `Branding` class (component) with this new `MyBranding` class (component).
- * The rest of the code is related the content and styling of the component.

Now, you can run the application to see the result:

```
![bookstore-added-logo](../../images/bookstore-added-logo.png)
```

➤ Since the component inherits from the component it is replacing, you can use all the non-private fields/properties/methods of the base component in the derived component.

Example: Replacing with the Code Behind File

If you prefer to use code-behind file for the C# code of your component, you can use the attributes in the C# side.

****MyBlazor.razor****

```
```html
{{if UI == "BlazorServer"}}
@using Volo.Abp.AspNetCore.Components.Server.BasicTheme.Themes.Basic
{{end}}
{{if UI == "Blazor"}}
@using Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme.Themes.Basic
{{end}}
@inherits Branding


```
```

****MyBlazor.razor.cs****

```
```csharp
{{if UI == "BlazorServer"}}
using Volo.Abp.AspNetCore.Components.Server.BasicTheme.Themes.Basic;
{{end}}
{{if UI == "Blazor"}}
using Volo.Abp.AspNetCore.Components.WebAssembly.BasicTheme.Themes.Basic;
{{end}}

using Volo.Abp.DependencyInjection;

namespace MyProject.Blazor.Components
{
 [ExposeServices(typeof(Branding))]
 [Dependency(ReplaceServices = true)]
 public partial class MyBranding
 {

 }
}
```
```

Theming

The [\[Theming\]](#) (`Theming.md`) system allows you to build your own theme. You can create your theme from scratch or get the [\[Basic Theme\]](#) (`Basic-Theme.md`) and change however you like.

10.2.11 Global Scripts & Styles

Blazor UI: Managing Global Scripts & Styles

Some modules may require additional styles or scripts that need to be referenced in ****index.html**** file. It's not easy to find and update these types of references in Blazor

apps. ABP offers a simple, powerful, and modular way to manage global style and scripts in Blazor apps.

To update script & style references without worrying about dependencies, ordering, etc in a project, you can use the [bundle command](../../CLI.md#bundle).

You can also add custom styles and scripts and let ABP manage them for you. In your Blazor project, you can create a class implementing `IBundleContributor` interface.

`IBundleContributor` interface contains two methods.

- * `AddScripts(...)`
- * `AddStyles(...)`

Both methods get `BundleContext` as a parameter. You can add scripts and styles to the `BundleContext` and run [bundle command](../../CLI.md#bundle). Bundle command detects custom styles and scripts with module dependencies and updates `index.html` file.

```
## Example Usage
```csharp
namespace MyProject.Blazor
{
 public class MyProjectBundleContributor : IBundleContributor
 {
 public void AddScripts(BundleContext context)
 {
 context.Add("site.js");
 }

 public void AddStyles(BundleContext context)
 {
 context.Add("main.css");
 context.Add("custom-styles.css");
 }
 }
}
```

```

➤ There is a BundleContributor class implementing `IBundleContributor` interface coming by default with the startup templates. So, most of the time, you don't need to add it manually.

Bundling And Minification

`abp bundle` command offers bundling and minification support for client-side resources(JavaScript and CSS files). `abp bundle` command reads the `appsettings.json` file inside the Blazor project and bundles the resources according to the configuration. You can find the bundle configurations inside `AbpCli.Bundle` element.

Here are the options that you can control inside the `appsettings.json` file.

`Mode` : Bundling and minification mode. Possible values are

- * `BundleAndMinify` : Bundle all the files into a single file and minify the content.

* `Bundle` : Bundle all files into a single file, but not minify.
* `None` : Add files individually, do not bundle.

`Name` : Bundle file name. Default value is `global`.

`Parameters` : You can define additional key/value pair parameters inside this section. `abp bundle` command automatically sends these parameters to the bundle contributors, and you can check these parameters inside the bundle contributor, take some actions according to these values.

Let's say that you want to exclude some resources from the bundle and control this action using the bundle parameters. You can add a parameter to the bundle section like below.

```
```json
"AbpCli": {
 "Bundle": {
 "Mode": "BundleAndMinify", /* Options: None, Bundle, BundleAndMinify */
 "Name": "global",
 "Parameters": {
 "ExcludeThemeFromBundle": "true"
 }
 }
}
```

```

You can check this parameter and take action like below.

```
```csharp
public class MyProjectNameBundleContributor : IBundleContributor
{
 public void AddScripts(BundleContext context)
 {
 }

 public void AddStyles(BundleContext context)
 {
 var excludeThemeFromBundle =
bool.Parse(context.Parameters.GetValueOrDefault("ExcludeThemeFromBundle"));
 context.Add("mytheme.css", excludeFromBundle: excludeThemeFromBundle);
 context.Add("main.css");
 }
}
```

```

10.2.12 Global Features

Blazor UI: Global Features

`GlobalFeatureManager` allows you to check the global features in your Blazor applications.

```

## Usage

```html
@using Volo.Abp.GlobalFeatures

@* ... *@

*@ Global Feature can be checked with feature name *@
@if(GlobalFeatureManager.Instance.IsEnabled("Ecommerce.Subscription"))
{
 Ecommerce.Subscription is enabled.
}

*@ OR it can be checked with type *@

@if(GlobalFeatureManager.Instance.IsEnabled<CommerceSubscriptionGlobalFeature>())
{
 Ecommerce.Subscription is enabled.
}
```

```

- You can follow *_Check for a Global Feature_* section of the [Global Features document](../../Global-Features.md) to check global features in your C# code.

10.2.13 Routing

Blazor UI: Routing

Blazor has its own [routing system](https://docs.microsoft.com/en-us/aspnet/core/blazor/fundamentals/routing) and you can use it in your applications. ABP doesn't add any new feature to it, except one small improvement for the [modular development](../../Module-Development-Basics.md).

AbpRouterOptions

Blazor `Router` component requires to define `AdditionalAssemblies` when you have components in assemblies/projects other than the main application's entrance assembly. So, if you want to create razor class libraries as ABP modules, you typically want to add the module's assembly to the `AdditionalAssemblies`. In this case, you need to add your module's assembly to the `AbpRouterOptions`.

****Example****

```

```csharp
Configure<AbpRouterOptions>(options =>
{
 options.AdditionalAssemblies.Add(typeof(MyBlazorModule).Assembly);
});
```

```

Write this code in the `ConfigureServices` method of your [module](../../Module-Development-Basics.md).

`AbpRouterOptions` has another property, `AppAssembly`, which should be the entrance assembly of the application and typically set in the final application's module. If you've created your solution with the [application startup template](../../Startup-Templates/Application.md), it is already configured for you.

See Also

- * [Blazor Routing](https://docs.microsoft.com/en-us/aspnet/core/blazor/fundamentals/routing) (Microsoft Documentation)

10.2.14 PWA Configuration

PWA Configuration

[PWAs (Progressive Web Apps)](https://web.dev/progressive-web-apps/) are developed using specific technologies to allow applications to take advantage of both web and native app features.

Here is a list of some features that PWA provides:

- **Installable**: A web application can be installed and used like a native/desktop application.
- **Network Independent**: PWAs support offline scenarios. It can work offline or with a poor network connection.
- **Responsive**: It's usable on any devices such as mobile phones, tablets, laptops, etc.

Creating a Project with PWA Support

You can create a new web application with PWA support for **Blazor WebAssembly** by using the `--pwa` option as below:

```
```bash
abp new Acme.BookStore -t blazor --pwa
```
```

After this command, your application will be created and some additional PWA related files (such as **manifest**, **icons**, **service workers**, etc.) will be added. Then, you can get the full advantages of web and native app features.

Adding PWA Support to an Existing Project

If you started your application without PWA support, it's possible to change your mind and get the benefit of PWA later. You only need to make some configurations as listed below:

1-) Add the `manifest.json` File

- Web Application Manifest provides information about a web application in a JSON text file and it's required for the web application to be downloaded and be presented to the user similarly to a native application.

First, you need to create a JSON file named **manifest.json** under the **wwwroot** folder and define some pieces of information about your application. You can see an example **manifest.json** file content below:

```
```json
{
 "name": "MyProjectName",
 "short_name": "MyCompanyName.MyProjectName",
 "start_url": "./",
 "display": "standalone",
 "background_color": "#ffffff",
 "theme_color": "#03173d",
 "prefer_related_applications": false,
 "icons": [
 {
 "src": "icon-512.png",
 "type": "image/png",
 "sizes": "512x512"
 },
 {
 "src": "icon-192.png",
 "type": "image/png",
 "sizes": "192x192"
 }
]
}
```

```

- Some application specific information should be defined in this file.
- For example, you can configure which icon needs to be seen in which screen size, background color, description, etc.

2-) Add Icons for Specific Screen Sizes (Optional)

You can add some icons for your application to be seen in specific screen sizes and define in which screen sizes icons should be displayed in the **manifest.json** file. You can see the **icons** section in the **manifest.json** file as an example above.

- You can use, default icons from our [template](<https://github.com/abpframework/abp/tree/dev/templates/app/aspnet-core/src/MyCompanyName.MyProjectName.Blazor/wwwroot>).

3-) Configure Service Workers

- Service workers are one of the fundamental parts of PWAs. They enable fast loading (regardless of the network), offline access, push notifications, and other web/native app capabilities. They run in the background and don't block the main thread so they don't slow your application.

You need to create `service-worker.js` and `service-worker.published.js` files under the **wwwroot** folder of your project. These files will be used by your project to determine which PWA features you want to use.

You can get the simple configurations for the [service-worker.js](<https://github.com/abpframework/abp/blob/dev/templates/app/aspnet-core/src/MyCompanyName.MyProjectName.Blazor/wwwroot/service-worker.js>) and [service-worker.published.js](<https://github.com/abpframework/abp/blob/dev/templates/app/aspnet-core/src/MyCompanyName.MyProjectName.Blazor/wwwroot/service-worker.published.js>) files from our [template](<https://github.com/abpframework/abp/tree/dev/templates/app/aspnet-core/src/MyCompanyName.MyProjectName.Blazor/wwwroot>).

After the related service worker files are added, then we need to define them in our `*.csproj` file to notify our application. So open your `*.csproj` file and add the following content:

```
```xml
<PropertyGroup>
 <TargetFramework>net6.0</TargetFramework>

 <BlazorWebAssemblyLoadAllGlobalizationData>true</BlazorWebAssemblyLoadAllGlobalizationData>

 <!-- Add the following line -->
 <ServiceWorkerAssetsManifest>service-worker-assets.js</ServiceWorkerAssetsManifest>
</PropertyGroup>

<!-- Add the following item group -->
<ItemGroup>
 <ServiceWorker Include="wwwroot\service-worker.js" PublishedContent="wwwroot\service-worker.published.js" />
</ItemGroup>
```

```

- * With the `ServiceWorkerAssetsManifest` MSBuild property, your Blazor application generates a service worker assets manifest with the specified name. This file will be generated in the path of `/bin/Debug/{TARGET FRAMEWORK}/wwwroot/service-worker-assets.js` on runtime. This manifest can list all resources such as images, stylesheets, JS files etc. by examining the `service-worker.published.js` file (regarding to your configurations in this file).
- * The `ServiceWorker` property is used to define which files need to be accounted as **Service Worker** files and service workers are used to determine which PWA features should be used.

4-) Define Web Application Manifest and Register Service Workers

Finally, now you can define the `manifest.json` file and **icons** in the **index.html** file and register the **service workers** for your application.

Let's start with adding `<link>` elements (between `<head>` tags) for the manifest and app icon in the **index.html** file (under the **wwwroot** folder):

```
```html
<head>
 <!-- ... -->
 <link href="manifest.json" rel="manifest" />
 <link rel="apple-touch-icon" sizes="512x512" href="icon-512.png" />
 <link rel="apple-touch-icon" sizes="192x192" href="icon-192.png" />
</head>
```
```

Then, add the following `<script>` tag inside the closing `</body>` tag in the same file:

```
```html
<body>
 <!-- ... -->
 <script>
 if ('serviceWorker' in navigator) {
 navigator.serviceWorker.register("service-worker.js");
 }
 </script>
</body>
```
```

You've added the related files and made the related configurations with this final touch to add PWA support to your existing application. Now, you can take full advantage of PWAs.

Differences Between the Debug and Published Service Workers

Application Template produces two service worker files, if you create your application with PWA support:

- * The `service-worker.js` file is used during development and does nothing by default.
- * The `service-worker.published.js` file, which is used after the app is published. Caches certain file extensions and supports offline scenarios by default (uses a *cache-first* strategy). A user must first visit the app while they're online. The browser automatically downloads and caches all of the resources required to operate offline and then when the network connection is disconnected, it can be used like before.

You can configure those files as mentioned in the *Customize Service Workers* section down below.

- If you want to share logic between those two service worker files, you can consider creating a third JS file and hold the common logic in this file or use the `[self.importScripts](https://developer.mozilla.org/docs/Web/API/WorkerGlobalScope/importScripts)`s to load the common logic into both service worker files.

Customization

You can customize the `manifest.json`, `service-worker.js` and `service-worker.published.js` files generated by the ABP Framework if you created an application with PWA support.

Customize Web Application Manifest (`manifest.json`)

➤ The web app manifest is a JSON file that tells the browser about your Progressive Web App and how it should behave when installed on the user's desktop or mobile device. A typical manifest file includes the app name, the icons the app should use, and the URL that should be opened when the app is launched. – From [web.dev](<https://web.dev/add-manifest>)

You can customize the `manifest.json` file (under the **wwwroot** folder) to your needs. You can set the **name**, **short_name**, **icons**, **description**, **start_url**, etc. You can see an example `manifest.json` file content below:

```
```json
{
 "name": "Acme.BookStore",
 "short_name": "BookStore",
 "description": "My application description",
 "theme_color": "#000000",
 "background_color": "#ffffff",
 "icons": [
 {
 "src": "../icon-192.png",
 "sizes": "192x192",
 "type": "image/png"
 },
 {
 "src": "../icon-512.png",
 "sizes": "512x512",
 "type": "image/png"
 }
]
 // other properties...
}
```

```

* You must provide at least the `short_name` or `name` property. If both of these properties are provided, the `short_name` property is used almost anywhere like the **launcher** and the **home** screen.

* For Chromium based browsers, you must provide at least a *192x192* px icon and a *512x512* px icon. If only those two icon sizes are provided, the browsers will automatically scale the icons to fit the device. If you don't want to let the browser auto-scale icons, you need to add icons for other sizes too.

➤ You can see the other properties from [here](<https://web.dev/add-manifest/#manifest-properties>).

Customize Service Workers

If you create your application with PWA support, two service worker files will be generated: `service-worker.js` and `service-worker.published.js`.

ABP Framework's service-worker files are the same as the .NET Core's and it's valid for most of the time and you'll probably not need to configure it manually. However, if you want to configure the service workers you can do it easily.

You can configure the `service-worker.js` file for debug mode and the `service-worker.published.js` file for release mode according to your own needs.

```
#### `service-worker.js`
```

```
```js
// Caution: In development, always fetch from the network and do not enable offline
support.
self.addEventListener('fetch', () => { });
````
```

* Configuring this file, you can use additional PWA features in debug mode.

* By default, it does nothing in debug mode, it fetches from the network (recommended) and does not support offline scenarios. You can change this behavior by configuring this file and also benefit from additional features of PWAs.

```
#### `service-worker.published.js`
```

```
```js
// Caution: Be sure you understand the caveats before publishing an application with
// offline support. See https://aka.ms/blazor-offline-considerations

self.importScripts('./service-worker-assets.js');
self.addEventListener('install', event => event.waitUntil(onInstall(event)));
self.addEventListener('activate', event => event.waitUntil(onActivate(event)));
self.addEventListener('fetch', event => event.respondWith(onFetch(event)));

const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const offlineAssetsInclude = [/\.dll$/, /\.pdb$/, /\.wasm$/, /\.html$/, /\.js$/, /\.json$/,
/\.css$/, /\.woff$/, /\.png$/g, /\.jpe?g$/g, /\.gif$/g, /\.ico$/g, /\.blat$/g, /\.dat$/g];
const offlineAssetsExclude = [/^service-worker\.js$/];

async function onInstall(event) {
 console.info('Service worker: Install');

 // Fetch and cache all matching items from the assets manifest
 const assetsRequests = self.assetsManifest.assets
 .filter(asset => offlineAssetsInclude.some(pattern => pattern.test(asset.url)))
 .filter(asset => !offlineAssetsExclude.some(pattern => pattern.test(asset.url)))
 .map(asset => new Request(asset.url, { integrity: asset.hash, cache: 'no-
cache' }));
 await caches.open(cacheName).then(cache => cache.addAll(assetsRequests));
}
```

```

async function onActivate(event) {
 console.info('Service worker: Activate');

 // Delete unused caches
 const cacheKeys = await caches.keys();
 await Promise.all(cacheKeys
 .filter(key => key.startsWith(cacheNamePrefix) && key !== cacheName)
 .map(key => caches.delete(key)));
}

async function onFetch(event) {
 let cachedResponse = null;
 if (event.request.method === 'GET') {
 // For all navigation requests, try to serve index.html from cache
 // If you need some URLs to be server-rendered, edit the following check to
 // exclude those URLs
 const shouldServeIndexHtml = event.request.mode === 'navigate';

 const request = shouldServeIndexHtml ? 'index.html' : event.request;
 const cache = await caches.open(cacheName);
 cachedResponse = await cache.match(request);
 }

 return cachedResponse || fetch(event.request);
}
```

```

- * You can configure this file if you want to cache additional file extensions such as `.`.webp` or etc. You can also use some additional features of PWA by configuring this file.
- * By default, dll files (`*.dll`) and some static assets (`*.js`, `*.css`, etc.) are cached.
- * Cached files will be stored in the `service-worker-assets.js` (**/bin/Debug/{TARGET FRAMEWORK}/wwwroot/service-worker-assets.js**). You can change this file name by renaming it in between the `ServiceWorkerAssetsManifest` tags on your `*.csproj` file.

See Also

- * [ASP.NET Core Blazor Progressive Web Application (PWA)](https://docs.microsoft.com/en-us/aspnet/core/blazor/progressive-web-app).

10.2.15 Layout Hooks

Blazor UI: Layout Hooks

ABP Framework theming system places the page layout into the [theme] (Theming.md) NuGet packages. That means the final application doesn't include a layout, so you can't directly change the layout code to customize it.

- > If you create a Blazor WASM project, the `index.html` file will be included within the template. You can also customize it to your needs.

You can copy the theme code into your solution. In this case, you are completely free to customize it. However, then you won't be able to get automatic updates of the theme (by upgrading the theme NuGet package).

The **Layout Hook System** allows you to **add code** at some specific parts of the layout. All layouts of all themes should implement these hooks. Finally, you can add a **razor component** into a hook point.

Example: Add a Simple Announcement Alert

Assume that you need to add a simple banner to the layout (that will be available for all the pages) to make an announcement about your new product. First, **create a razor component** in your project:

```
![bookstore-banner-component](../../images/bookstore-banner-component.png)

**AnnouncementComponent.razor.cs**

```csharp
using System.Threading.Tasks;
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace Acme.BookStore.Blazor.Components;

public partial class AnnouncementComponent : ComponentBase
{
 private const string AnnouncementLocalStorageKey = "product-announcement-status";

 [Inject]
 public IJSRuntime JsRuntime { get; set; }

 public bool ShowBanner { get; set; }

 protected override async Task OnInitializedAsync()
 {
 ShowBanner = await ShowAnnouncementBannerAsync();
 }

 private async Task<bool> ShowAnnouncementBannerAsync()
 {
 var announcementLocalStorageValue = await
JsRuntime.InvokeAsync<string>("localStorage.getItem", AnnouncementLocalStorageKey);

 return announcementLocalStorageValue != null &&
 bool.TryParse(announcementLocalStorageValue, out var
showAnnouncementBanner) && showAnnouncementBanner;
 }

 private async Task HideAnnouncementBannerAsync()
 {
 }
}
```

```

 await JsRuntime.InvokeVoidAsync("localStorage.setItem",
AnnouncementLocalStorageKey, true);
 ShowBanner = false;
 StateHasChanged();
 }
}
```

```

AnnouncementComponent.razor

```

```html
@if(ShowBanner)
{
 <div class="alert alert-info">
 A brand new product is in sale. Click here to learn more.
 <button @onclick="async () => { await
HideAnnouncementBannerAsync(); }">Hide</button>
 </div>
}
```

```

Then, you can add this component to any of the hook points in the `ConfigureServices` of your module class:

```

```csharp
Configure<AbpLayoutHookOptions>(options =>
{
 options.Add(
 LayoutHooks.Body.Last, //The hook name
 typeof(AnnouncementComponent) //The component to add
);
})
```

```

Now, the `AnnouncementComponent` will be rendered in the `body` of the page as the last item.

Specifying the Layout

The configuration above adds the `AnnouncementComponent` to all layouts. You may want to only add it to a specific layout:

```

```csharp
Configure<AbpLayoutHookOptions>(options =>
{
 options.Add(
 LayoutHooks.Body.Last,
 typeof(AnnouncementComponent),
 layout: StandardLayouts.Application //Set the layout to add
);
})
```

```

See the **Layouts** section below to learn more about the layout system.

Layout Hook Points

There are some pre-defined layout hook points. The standard hook points are:

- * `LayoutHooks.Body.First` : Used to add a component as the first item in the HTML body tag.
 - * `LayoutHooks.Body.Last` : Used to add a component as the last item in the HTML body tag.
- > You (or the modules you are using) can add **multiple items to the same hook point**. All of them will be added to the layout in the order they were added.

Layouts

The layout system allows themes to define the standard named layouts and allows any page to select a proper layout for its purpose. There is one pre-defined layout:

- * **Application**: The main (and the default) layout for an application. It typically contains a header, menu (sidebar), footer, toolbar... etc.

This layout is defined in the `StandardLayouts` class as constants. You can definitely create your own layouts, but this layout is the standard layout and it's implemented by all the themes out of the box.

- > If you don't specify the layout, your razor component will be rendered in all of the layouts.

Layout Location

You can find the `MainLayout.razor` [here](<https://github.com/abpframework/abp/blob/dev/modules/basic-theme/src/Volo.Abp.AspNetCore.Components.Web.BasicTheme/Themes/Basic/MainLayout.razor>) for the basic theme. You can take it as a reference to build your own layouts or you can override it, if necessary.

See Also

- * [Customization / Overriding Components] (Customization-Overriding-Components.md)

10.3 Angular

10.3.1 Quick Start

ABP Angular Quick Start

How to Prepare Development Environment

Please follow the steps below to prepare your development environment for Angular.

1. **Install Node.js:** Please visit [[Node.js downloads page](#)] (<https://nodejs.org/en/download/>) and download proper Node.js v16 or v18 installer for your OS. An alternative is to install [[NVM](#)] (<https://github.com/nvm-sh/nvm>) and use it to have multiple versions of Node.js in your operating system.
2. **[Optional] Install Yarn:** You may install Yarn v1 (not v2) following the instructions on [[the installation page](#)] (<https://classic.yarnpkg.com/en/docs/install>). Yarn v1 delivers an arguably better developer experience compared to npm v6 and below. You may skip this step and work with npm, which is built-in in Node.js, instead.
3. **[Optional] Install VS Code:** [[VS Code](#)] (<https://code.visualstudio.com/>) is a free, open-source IDE which works seamlessly with TypeScript. Although you can use any IDE including Visual Studio or Rider, VS Code will most likely deliver the best developer experience when it comes to Angular projects. ABP project templates even contain plugin recommendations for VS Code users, which VS Code will ask you to install when you open the Angular project folder. Here is a list of recommended extensions:
 - [[Angular Language Service](#)] (<https://marketplace.visualstudio.com/items?itemName=angular.ng-template>)
 - [[Prettier - Code formatter](#)] (<https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode>)
 - [[TSLint](#)] (<https://marketplace.visualstudio.com/items?itemName=ms-vscode.vscode-typescript-tslint-plugin>)
 - [[Visual Studio IntelliCode](#)] (<https://marketplace.visualstudio.com/items?itemName=visualstudioexptteam.vscode-deintelligicode>)
 - [[Path Intellisense](#)] (<https://marketplace.visualstudio.com/items?itemName=christian-kohler.path-intellisense>)
 - [[npm Intellisense](#)] (<https://marketplace.visualstudio.com/items?itemName=christian-kohler.npm-intellisense>)
 - [[Angular 10 Snippets - TypeScript, Html, Angular Material, ngRx, RxJS & Flex Layout](#)] (<https://marketplace.visualstudio.com/items?itemName=Mikael.Angular-BeastCode>)
 - [[JavaScript \(ES6\) code snippets](#)] (<https://marketplace.visualstudio.com/items?itemName=xabikos.JavaScriptSnippets>)
 - [[Debugger for Chrome](#)] (<https://marketplace.visualstudio.com/items?itemName=msjsdiag.debugger-for-chrome>)
 - [[Git History](#)] (<https://marketplace.visualstudio.com/items?itemName=donjayamanne.githistory>)
 - [[indent-rainbow](#)] (<https://marketplace.visualstudio.com/items?itemName=oderwat.indent-rainbow>)

How to Start a New Angular Project

You have multiple options to initiate a new Angular project that works with ABP:

1. Using ABP CLI

ABP CLI is probably the most convenient and flexible way to initiate an ABP solution with an Angular frontend. Simply [[install the ABP CLI](#)] ([./CLI.md](#)) and run the following command in your terminal:

```
```shell
abp new MyCompanyName.MyProjectName -csf -u angular
```
```

› To see further options in the CLI, please visit the [CLI manual](../../CLI.md).

This command will prepare a solution with an Angular and a .NET Core project in it. Please visit [Getting Started section](../../Getting-Started.md?UI=NG&DB=EF&Tiered=No#abp-cli-commands-options) for further instructions on how to set up the backend of your solution.

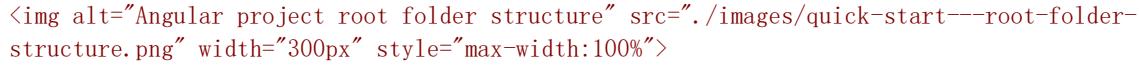
To continue reading without checking other methods, visit [Angular project structure section](#angular-project-structure).

2. Generating a CLI Command from Get Started Page

You can generate a CLI command on the [get started page of the abp.io website](https://abp.io/get-started). Then, use the command on your terminal to create a new [Startup Template](../../Startup-Templates/Index.md).

Angular Project Structure

After creating a solution, open its "angular" directory in your IDE. This is how the contents of the root folder looks like:

A screenshot showing the file structure of an Angular project. The root folder contains several subfolders: .vscode, e2e, src, browserlistrc, editorconfig, gitignore, prettier, angular.json, karma.conf.js, package.json, and README.md. Each folder has its own substructure, such as .vscode containing settings and extensions, and src containing components, services, and other Angular files.

Here is what these folders and files are for:

- ****.vscode**** has extension recommendations in it.
- ****e2e**** is a separate app for possible end-to-end tests.
- ****src**** is where the source files for your application are placed. We will have a closer look in a minute.
- ****.browserlistrc**** helps [configuring browser compatibility of your Angular app](https://angular.io/guide/build#configuring-browser-compatibility).
- ****.editorconfig**** helps you have a shared coding style for separate editors and IDEs. Check [EditorConfig.org](https://editorconfig.org/) for details.
- ****.gitignore**** defined which files and folders should not be tracked by git. Check [git documentation](https://git-scm.com/docs/gitignore) for details.
- ****.prettierrc**** includes simple coding style choices for [Prettier](https://prettier.io/), an auto-formatter for TypeScript, HTML, CSS, and more. If you install recommended extensions to VS Code, you will never have to format your code anymore.
- ****angular.json**** is where Angular workspace is defined. It holds project configurations and workspace preferences. Please refer to [Angular workspace configuration](https://angular.io/guide/workspace-config) for details.
- ****karma.conf.js**** holds [Karma test runner](https://karma-runner.github.io/) configurations.
- ****package.json**** is where your [package dependencies](https://angular.io/guide/npm-packages) are listed. It also includes some useful scripts for developing, testing, and building your application.
- ****README.md**** includes some of Angular CLI command examples. You either have to install Angular CLI globally or run these commands starting with `yarn` or `npx` to make them work.

- **start.ps1** is a simple PowerShell script to install dependencies and start a [development server via Angular CLI] (<https://angular.io/cli/serve>), but you probably will not need that after reading this document.
- **tsconfig.json** and all other [tsconfig files] (<https://angular.io/guide/typescript-configuration>) in general, include some TypeScript and Angular compile options.
- **yarn.lock** enables installing consistent package versions across different devices so that working application build will not break because of a package update. Please read [Yarn documentation] (<https://classic.yarnpkg.com/en/docs/yarn-lock/>) if you are interested in more information on the topic. If you have decided to use npm, please remove this file and keep the [package-lock.json] (<https://docs.npmjs.com/files/package-lock.json>) instead.

Now let us take a look at the contents of the source folder.

![Angular project source folder structure](./images/quick-start---source-folder-structure.png)

- **app** is the main directory you put your application files in. Any module, component, directive, service, pipe, guard, interceptor, etc. should be placed here. You are free to choose any folder structure, but [organizing Angular applications based on modules] (<https://angular.io/guide/module-types>) is generally a fine practice.
- **home** is a predefined module and acts as a welcome page. It also demonstrates how a feature-based folder structure may look like. More complex features will probably have sub-features, thus inner folders. You may change the home folder however you like.
- **shared** is spared for reusable code that works for several modules. Some, including yours truly, may disagree with using a single module for all shared code, so consider adding standalone sub-modules inside this folder instead of adding everything into **shared.module.ts**.
- **app-routing.module.ts** is where your top-level routes are defined. Angular is capable of [lazy loading feature modules] (<https://angular.io/guide/lazy-loading-ngmodules>), so not all routes will be here. You may think of Angular routing as a tree and this file is the top of the tree.
- **app.component.ts** is essentially the top component that holds the dynamic application layout.
- **app.module.ts** is the [root module] (<https://angular.io/guide/bootstrapping>) that includes information about how parts of your application are related and what to run at the initiation of your application.
- **route.provider.ts** is used for [modifying the menu] (<https://docs.abp.io/en/abp/latest/UI/Angular/Modifying-the-Menu>).
- **assets** is for static files. A file (e.g. an image) placed in this folder will be available as is when the application is served.
- **environments** includes one file per environment configuration. There are two configurations by default, but you may always introduce another one. These files are directly referred to in `_angular.json` and help you have different builds and application variables. Please refer to [configuring Angular application environments] (<https://angular.io/guide/build#configuring-application-environments>) for details.
- **index.html** is the HTML page served to visitors and will contain everything required to run your application. Servers should be configured to redirect every request to this page so that the Angular router can take over. Do not worry about how to add JavaScript and CSS files to it, because Angular CLI will do it automatically.
- **main.ts** bootstraps and configures Angular application to run in the browser. It is production-ready, so forget about it.

- **polyfill.ts** is where you can add polyfills if you want to [support legacy browsers](<https://angular.io/guide/browser-support>).
- **style.scss** is the default entry point for application styles. You can change this or add new entry points in `_angular.json`.
- **test.ts** helps the unit test runner discover and bootstrap spec files.

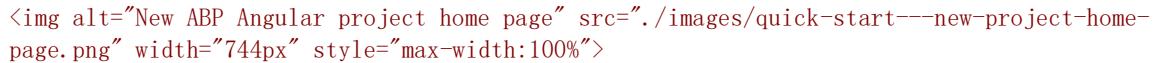
Phew! So many files, right? Yet, **most of them are typically not subject to change** or, even when they are so, the CLI tooling will do the job for you. The main focus should be on the app folder and its content.

Next, we will take a look at the commands used to prepare, build, and serve our application.

How to Run the Angular Application

Now that you know about the files and folders, we can get the application up and running.

1. Make sure the [database migration is complete]([../../Getting-Started?UI=NG&DB=EF&Tiered=No#create-the-database](#)) and the [API is up and running]([../../Getting-Started?UI=NG&DB=EF&Tiered=No#run-the-application](#)).
2. Run `yarn` or `npm install` if you have not already.
3. Run `yarn start` or `npm start`. The first compilation may take a while. This will start a [live development server]([#angular-live-development-server](#)) and launch your default browser in the end.
4. Visit the browser page that opens after the compilation `^{[1] (#f-certificate-error)}`.

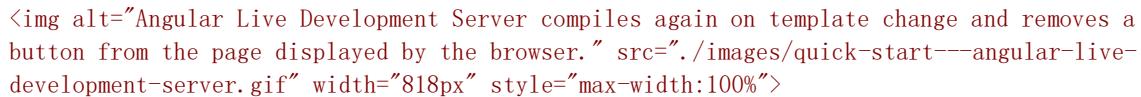


The image shows a screenshot of a web browser displaying the home page of a new ABP Angular project. The page has a clean, modern design with a light blue header and a white background. It features a large title 'Angular Project' and some descriptive text about the project's purpose and setup.

You may modify the behavior of the **start script** (in the package.json file) by changing the parameters passed to the `ng serve` command. For instance, if you do not want a browser window to open next time you run the script, remove `--open` from the end of it. Please check [ng serve documentation](<https://angular.io/cli/serve>) for all available options.

Angular Live Development Server

The development server of Angular is based on [Webpack DevServer](<https://webpack.js.org/configuration/dev-server/>). It tracks changes to source files and syncs the browser window after an incremental re-compilation every time `^{[2] (#f-dev-server)}` you make one. Your experience will be like this:



The image shows a screenshot of a web browser displaying the Angular Live Development Server. The page contains a single button labeled 'Click me'. A tooltip appears over the button, stating 'Compiling...'. After a brief delay, the button disappears, and a message 'Template compiled successfully.' is displayed below it.

Please keep in mind that you should not use this server in production. To provide the fastest experience, the compiler skips some heavy optimizations and the development server is simply not built for multiple clients. The next section will describe what to do.



^{1} _If you see the error above when you run the Angular app, your browser might be blocking access to the API because of the self-signed certificate. Visit that address and allow access to it (once). When you see the Swagger interface, you are good to go._ **^{[[←](#)] (#a-certificate-error)}**

^{2} _Sometimes, depending on the file changed, Webpack may miss the change and cannot reflect it in the browser. For example, tsconfig files are not being tracked. In such a case, please restart the development server._ **^{[[←](#)] (#a-dev-server)}**

How to Build the Angular Application

An Angular application can have multiple [build targets](<https://angular.io/guide/glossary#target>), i.e. **configurations in angular.json** which define how [Architect](<https://angular.io/guide/glossary#architect>) will build applications and libraries. Usually, each build configuration has a separate environment variable file. Currently, the project has two: One for development and one for production.

```
```js
// this is what environment variables look like
// can be found at /src/environments/environment.ts

import { Config } from '@abp/ng.core';

const baseUrl = 'http://localhost:4200';

export const environment = {
 production: false,
 application: {
 baseUrl,
 name: 'MyProjectName',
 logoUrl: '',
 },
 oAuthConfig: {
 issuer: 'https://localhost:44381',
 redirectUri: baseUrl,
 clientId: 'MyProjectName_App',
 responseType: 'code',
 scope: 'offline_access MyProjectName',
 },
 apis: {
 default: {
 url: 'https://localhost:44381',
 rootNamespace: 'MyCompanyName.MyProjectName',
 }
 }
};
```

```
 },
},
} as Config.Environment;
```

```

When you run the development server, variables defined in `_environment.ts` take effect. Similarly, in production mode, the default environment is replaced by `_environment.prod.ts` and completely different variables become effective. You may even [create a new build configuration](<https://angular.io/guide/workspace-config#build-configs>) and set [file replacements](<https://angular.io/guide/build#configure-target-specific-file-replacements>) to use a completely new environment. For now, we will start a production build:

1. Open your terminal and navigate to the root Angular folder.
2. Run `yarn` or `npm install` if you have not installed dependencies already.
3. Run `yarn build:prod` or `npm run build:prod`.



The image shows a screenshot of a terminal window with the Angular CLI command `ng build --prod` being run. The output shows the compilation process, including the generation of a self-signed certificate due to the use of the --prod flag. The terminal also shows the creation of a `dist` folder and the deployment of files to it.

Depending on project size, the compilation may take a few minutes. When it is finished, the compiled output will be placed inside the `./dist` folder. Voila! You have deployment-ready build artifacts.

➤ The amount of optimization performed on the source is the main difference between a regular build and a production one. Production builds have a much smaller size and are more performant.

How to Deploy the Angular Application

Angular web applications run on the browser and require no server except for a [static web server](https://developer.mozilla.org/en-US/docs/Learn/Common_questions/What_is_a_web_server) to deliver files to the client. To see that it works, please make sure the backend application is up and then run the following command in your terminal:

```
```shell
please replace MyProjectName with your project name

npx serve dist/MyProjectName index.html 4200 --browse
```

```

This command will download and start a simple static server, a browser window at `http://localhost:4200` will open, and the compiled output of your project will be served.

Of course, you need your application to run on an optimized web server and become available to everyone. This is quite straight-forward:

1. Create a new static web server instance. You can use a service like [Azure App Service](<https://azure.microsoft.com/en-us/services/app-service/web/>), [Firebase](<https://firebase.google.com/docs/hosting>), [Netlify](<https://www.netlify.com/>), [Vercel](<https://vercel.com/>), or even [GitHub

[Pages](#)](<https://angular.io/guide/deployment#deploy-to-github-pages>). Another option is maintaining own web server with [\[NGINX\]](#)(<https://www.nginx.com/>), [\[IIS\]](#)(<https://www.iis.net/>), [\[Apache HTTP Server\]](#)(<https://httpd.apache.org/>), or equivalent.

2. Copy the files from `dist/MyProjectName` ^[1] (#f-dist-folder-name) to a publicly served destination on the server via CLI of the service provider, SSH, or FTP (whichever is available). This step would be defined as a job if you have a CI/CD flow.
3. [\[Configure the server\]](#)(<https://angular.io/guide/deployment#server-configuration>) to redirect all requests to the _index.html_ file. Some services do that automatically. Others require you [\[to add a file to the bundle via assets\]](#)(<https://angular.io/guide/workspace-config#assets-configuration>) which describes the server how to do the redirections. Occasionally, you may need to do manual configuration.

In addition, you can [\[deploy your application to certain targets using the Angular CLI\]](#)(<https://angular.io/guide/deployment#automatic-deployment-with-the-cli>). Here are some deploy targets:

- [\[Azure\]](#)(<https://github.com/Azure/ng-deploy-azure#readme>)
- [\[Firebase\]](#)(<https://github.com/angular/angularfire#readme>)
- [\[Netlify\]](#)(<https://github.com/ngx-builders/netlify-builder#readme>)
- [\[Vercel `vercel init angular`\]](#)(<https://github.com/vercel/vercel/tree/main/examples/angular>)
- [\[GitHub Pages\]](#)(<https://github.com/angular-schule/angular-cli-ghpages/#readme>)

^{1} _The compiled output will be placed under `dist` in a folder by the project name._ ^[2] (#a-dist-folder-name) </sup></sup>

10.3.2 Development

10.3.2.1 Environment Variables

Environment

Every application needs some ****environment**** variables. In Angular world, this is usually managed by `environment.ts`, `environment.prod.ts` and so on. It is the same for ABP as well.

Current `Environment` configuration holds sub config classes as follows:

```
```js
export interface Environment {
 apis: Apis;
 application: Application;
 oAuthConfig: AuthConfig;
 production: boolean;
 remoteEnv?: RemoteEnv;
}
```

```

```
## Apis

```js
export interface Apis {
 [key: string]: ApiConfig;
 default: ApiConfig;
}

export interface ApiConfig {
 [key: string]: string;
 rootNamespace?: string;
 url: string;
}
```

```

Api config has to have a default config and it may have some additional ones for different modules.

I.e. you may want to connect to different Apis for different modules.

Take a look at following example

```

```json
{
 // ...
 "apis": {
 "default": {
 "url": "https://localhost:8080"
 },
 "AbpIdentity": {
 "url": "https://localhost:9090"
 }
 }
 // ...
}
```

```

When an api from `AbpIdentity` is called, the request will be sent to `https://localhost:9090`.

Everything else will be sent to `https://localhost:8080`

- `rootNamespace` ****(new)**** : Root namespace of the related API. e.g. Acme.BookStore

Application

```

```js
export interface Application {
 name: string;
 baseUrl?: string;
 logoUrl?: string;
}
```

```

```
```
```

- `name`: Name of the backend Application. It is also used by `logo.component` if `logoUrl` is not provided.
- `logoUrl`: Url of the application logo. It is used by `logo.component`.
- `baseUrl`: [For detailed information](./Multi-Tenancy.md#domain-tenant-resolver)

## ## AuthConfig

For authentication, we use angular-oauth2-oidc. Please check their [docs](<https://github.com/manfredsteyer/angular-oauth2-oidc>) out

## ## RemoteEnvironment

Some applications need to integrate an existing config into the `environment` used throughout the application.

Abp Framework supports this out of box.

To integrate an existing config json into the `environment`, you need to set `remoteEnv`

```
```js
export type customMergeFn = (
    localEnv: Partial<Config.Environment>,
    remoteEnv: any
) => Config.Environment;

export interface RemoteEnv {
    url: string;
    mergeStrategy: "deepmerge" | "overwrite" | customMergeFn;
    method?: string;
    headers?: ABP.Dictionary<string>;
}
```

- `url` *: Required. The url to be used to retrieve environment config
- `mergeStrategy` *: Required. Defines how the local and the remote `environment` json will be merged
 - `deepmerge`: Both local and remote `environment` json will be merged recursively. If both configs have same nested path, the remote `environment` will be prioritized.
 - `overwrite`: Remote `environment` will be used and local environment will be ignored.
 - `customMergeFn`: You can also provide your own merge function as shown in the example. It will take two parameters, `localEnv: Partial<Config.Environment>` and `remoteEnv` and it needs to return a `Config.Environment` object.
- `method`: HTTP method to be used when retrieving environment config. Default: `GET`
- `headers`: If extra headers are needed for the request, it can be set through this field.
```

## ## EnvironmentService

`EnvironmentService` is a singleton service, i.e. provided in root level of your application, and keeps the environment in the internal store.

#### #### Before Use

In order to use the `EnvironmentService` you must inject it in your class as a dependency.

```
```js
import { EnvironmentService } from '@abp/ng.core';

@Component({
  /* class metadata here */
})
class DemoComponent {
  constructor(private environment: EnvironmentService) {}
}
```

```

You do not have to provide the `EnvironmentService` at module or component/directive level, because it is already **\*\*provided in root\*\***.

#### #### Get Methods

`EnvironmentService` has numerous get methods which allow you to get a specific value or all environment object.

Get methods with "\$" at the end of the method name (e.g. `getEnvironment\$`) return an RxJs stream. The streams are triggered when set or patched the state.

#### ##### How to Get Environment Object

You can use the `getEnvironment` or `getEnvironment\$` method of `EnvironmentService` to get all of the environment object. It is used as follows:

```
```js
// this.environment is instance of EnvironmentService

const environment = this.environment.getEnvironment();

// or
this.environment.getEnvironment$().subscribe((environment) => {
  // use environment here
});
```

```

#### ##### How to Get API URL

The `getApiUrl` or `getApiUrl\$` method is used to get a specific API URL from the environment object. This is how you can use it:

```
```js
// this.environment is instance of EnvironmentService

const apiUrl = this.environment.getApiUrl();
// environment.apis.default.url
```

```

```
this.environment.getApiUrl$("search").subscribe((searchUrl) => {
 // environment.apis.search.url
});
```

This method returns the `url` of a specific API based on the key given as its only parameter. If there is no key, ``default`` is used.

#### #### How to Set the Environment

`EnvironmentService` has a method named `setState` which allows you to set the state value.

```
```js
// this.environment is instance of EnvironmentService

this.environment.setState(newEnvironmentObject);
```

```

Note that \*\*you do not have to call this method at application initiation\*\*, because the environment variables are already being stored at start.

#### #### Environment Properties

Please refer to `Environment` type for all the properties. It can be found in the [environment.ts file](<https://github.com/abpframework/abp/blob/dev/npm/ng-packs/packages/core/src/lib/models/environment.ts#L4>).

### 10.3.2.2 About Feature Libraries

#### # About Feature Libraries

ABP has an ever-growing number of feature modules and [introducing a new one]([./Module-Development-Basics.md](#)) is always possible. When the UI is Angular, these features have modular Angular libraries accompanying them.

#### ## Feature Library Content

Each library has at least two modules:

1. The main module contains all components, services, types, enums, etc. to deliver the required UI when the feature is loaded. From here on, we will refer to these modules as \*\*"feature module"\*\*.
2. There is also a \*\*"config module"\*\* per library which helps us configure applications to run these modules or make them accessible.

#### ## How to Add a Feature Library to Your Project

```
<!-- TODO: Insert info on CLI `add-module` command here when the schematic is ready. -->
```

The manual setup of a feature library has three steps:

#### ### 1. Install the Library

Feature libraries are usually published as an npm package. If a library you want to use does not exist in your project, you may install it via the following command:

```
```shell
yarn add @my-company-name/my-project-name
````
```

...or...

```
```shell
npm install @my-company-name/my-project-name
````
```

The `my-company-name` and `my-project-name` parts are going to change according to the package you want to use. For example, if we want to install the ABP Identity module, the package installation will be as seen below:

```
```shell
yarn add @abp/ng.identity
````
```

> Identity is used just as an example. If you have initiated your project with ABP CLI or ABP Suite, the identity library will already be installed and configured in your project.

#### ### 2. Import the Config Module

As of ABP v3.0, every lazy-loaded module has a config module available via a secondary entry point on the same package. Importing them in your root module looks like this:

```
```js
import { IdentityConfigModule } from "@abp/ng.identity/config";

@NgModule({
  imports: [
    // other imports
    IdentityConfigModule.forRoot(),
  ],
  // providers, declarations, and bootstrap
})
export class AppModule {}````
```

We need the config modules for actions required before feature modules are loaded (lazily). For example, the above import configures the menu to display links to identity pages.

Furthermore, depending on the library, the ` `.forRoot` static method may receive some options that configure how the feature works.

3. Import the Feature Module

Finally, the feature module should be [loaded lazily via Angular router](https://angular.io/guide/lazy-loading-ngmodules). If you open the `/src/app/app-routing.module.ts` file, you should see `IdentityModule` is loaded exactly as follows:

```
```js
import { NgModule } from "@angular/core";
import { RouterModule, Routes } from "@angular/router";

const routes: Routes = [
 // other routes
 {
 path: "identity",
 loadChildren: () =>
 import("@abp/ng.identity").then((m) => m.IdentityModule.forLazy()),
 },
 // other routes
];

@NgModule({
 imports: [RouterModule.forRoot(routes)],
 exports: [RouterModule],
})
export class AppRoutingModule {}```

```

When you load the identity feature like this, the "Users" page, for example, will have a route path of `/identity/users`. <sup>[1] (#f-modify-route)</sup>

Depending on the library, the ` `.forLazy` static method may also receive some options that configure how the feature works.

---

<sup><b>1</b></sup> \_Libraries expect to work at a predefined path. Please check [how to patch a navigation element](./Modifying-the-Menu.md#how-to-patch-or-remove-a-navigation-element), if you want to use a different path from the default one (e.g. '/identity').\_ <sup>[2]</sup> (#a-modify-route)

#### 10.3.2.3 Service Proxies

##### ## Service Proxies

Calling a REST endpoint from Angular applications is common. We usually create **services** matching server-side controllers and **interfaces** matching [DTOs](../../Data-Transfer-Objects) to interact with the server. This often results in

manually transforming C# code into TypeScript equivalents and that is unfortunate, if not intolerable.

To avoid manual effort, we might use a tool like [\[NSWAG\]](https://github.com/RicoSuter/NSwag) (<https://github.com/RicoSuter/NSwag>) that generates service proxies. However, NSWAG has some disadvantages:

- It generates **\*\*a single .ts file\*\*** which gets **\*\*too large\*\*** as your application grows. Also, this single file does not fit the **\*\*[modular](../../Module-Development-Basics) approach\*\*** of ABP.
- To be honest, the generated code is a bit **\*\*ugly\*\***. We would like to produce code that looks as if someone wrote it.
- Since `swagger.json` **\*\*does not reflect the exact method signature\*\*** of backend services, NSWAG cannot reflect them on the client-side as well.

ABP introduces an endpoint that exposes server-side method contracts. When the ``generate-proxy`` command is run, ABP CLI makes an HTTP request to this endpoint and generates better-aligned client proxies in TypeScript. It organizes folders according to namespaces, adds barrel exports, and reflects method signatures in Angular services.

› Before you start, please make sure you start the backend application with ``dotnet run``. There is a [\[known limitation about Visual Studio\]](#) (#known-limitations), so please do not run the project using its built-in web server.

Run the following command in the **\*\*root folder\*\*** of the angular application:

```
```bash
abp generate-proxy -t ng
````
```

The command without any parameters creates proxies only for your own application's services and places them in your default Angular application. There are several parameters you may use to modify this behavior. See the [\[CLI documentation\]](#) (../../CLI) for details.

The generated files will be placed in a folder called ``proxy`` at the root of the target project.

![generated-files-via-generate-proxy] (./images/generated-files-via-generate-proxy.png)

Each folder will have models, enums, and services defined at related namespace accompanied by a barrel export, i.e. an ``index.ts`` file for easier imports.

› The command can find application/library roots by reading the ``angular.json`` file. Make sure you have either defined your target project as the ``defaultProject`` or pass the ``--target`` parameter to the command. This also means that you may have a monorepo workspace.

### ### Angular Project Configuration

› If you've created your project with version 3.1 or later, you can skip this part since it will be already installed in your solution.

For a solution that was created before v3.1, follow the steps below to configure your Angular application:

1. Add `@abp/ng.schematics` package to the `devDependencies` of the Angular project. Run the following command in the root folder of the angular application:

```
```bash
npm install @abp/ng.schematics -D
````
```

2. Add `rootNamespace` property to the `/src/environments/environment.ts` in your application project as shown below. `MyCompanyName.MyProjectName` should be replaced by the root namespace of your .NET project.

```
```js
export const environment: Config.Environment = {
  // other environment variables...
  apis: {
    default: {
      rootNamespace: "MyCompanyName.MyProjectName",
      // other environment variables...
    },
  },
};
````
```

3. [OPTIONAL] Add the following paths to `tsconfig.base.json` in order to have a shortcut for importing proxies:

```
```json
{
  // other TS configuration...
  "compilerOptions": {
    // other TS configuration...
    "paths": {
      "@proxy": ["src/app/proxy/index.ts"],
      "@proxy/*": ["src/app/proxy/*"]
    }
  }
}
````
```

➤ The destination the `proxy` folder is created and the paths above may change based on your project structure.

### ### Services

The `generate-proxy` command generates one service per back-end controller and a method (property with a function value actually) for each action in the controller. These methods call backend APIs via [RestService](./Http-Requests#restservice).

A variable named `apiName` (available as of v2.4) is defined in each service. `apiName` matches the module's `RemoteServiceName`. This variable passes to the `RestService` as a parameter at each request. If there is no microservice API defined in the environment, `RestService` uses the default. See [getting a specific API endpoint from application config](./Http-Requests#how-to-get-a-specific-api-endpoint-from-application-config)

The `providedIn` property of the services is defined as ``root``. Therefore there is no need to provide them in a module. You can use them directly by injecting them into the constructor as shown below:

```
```js
import { BookService } from '@proxy/books';

@Component(/* component metadata here */)
export class BookComponent implements OnInit {
    constructor(private service: BookService) {}

    ngOnInit() {
        this.service.get().subscribe(
            // do something with the response
        );
    }
}
```

```

The Angular compiler removes the services that have not been injected anywhere from the final output. See the [tree-shakable providers documentation](<https://angular.io/guide/dependency-injection-providers#tree-shakable-providers>).

#### #### Models

The `generate-proxy` command generates interfaces matching DTOs in the back-end. There are also a few [core DTOs](<https://github.com/abpframework/abp/blob/dev/npm/ng-packs/packages/core/src/lib/models/dtos.ts>) in the `@abp/ng.core` package. In combination, these models can be used to reflect the APIs.

```
```js
import { PagedResultDto } from "@abp/ng.core";
import { BookDto } from "@proxy/books";

@Component(/* component metadata here */)
export class BookComponent implements OnInit {
    data: PagedResultDto<BookDto> = {
        items: [],
        totalCount: 0,
    };
}
```

```

#### #### Enums

Enums have always been difficult to populate in the frontend. The `generate-proxy` command generates enums in a separate file and exports a ready-to-use "options constant" from the same file. So you can import them as follows:

```
```js
import { bookGenreOptions } from "@proxy/books";

@Component(/* component metadata here */)
export class BookComponent implements OnInit {
    genres = bookGenreOptions;
}
```

```

...and consume the options in the template as follows:

```
```html
<!-- simplified for sake of clarity -->
<select formControlName="genre">
    <option [ngValue]="null">Select a genre</option>
    <option *ngFor="let genre of genres" [ngValue]="genre.value">
        {{ genre.key }}
    </option>
</select>
```

```

> Please [see this article](<https://github.com/abpframework/abp/blob/dev/docs/en/Blog-Posts/2020-09-07%20Angular-Service-Proxies/POST.md>) to learn more about service proxies.

#### ### Known Limitations

When you run a project on Visual Studio using IIS Express as the web server, there will be no remote access to your endpoints. This is the default behavior of IIS Express since it explicitly protects you from the security risks of running over the network. However, that will cause the proxy generator to fail because it needs a response from the `/api/abp/api-definition` endpoint. You may serve your endpoints via Kestrel to avoid this. Running `dotnet run` in your command line (at your project folder) will do that for you.

#### 10.3.2.4 PWA Configuration

##### # PWA Configuration

[Progressive Web Apps](<https://web.dev/progressive-web-apps/>) are web applications which, although not as integrated to the OS as a native app, can take advantage of native features. They can be discovered via search engines, installed on devices with a single tap or click, and shared via a regular link. They also can work offline and get updates when new content is available.

Converting your Angular application to a PWA is easy.

##### ## 1. Install Angular PWA

Run the following command in the root folder of your Angular application:

```
```shell
yarn ng add @angular/pwa
````
```

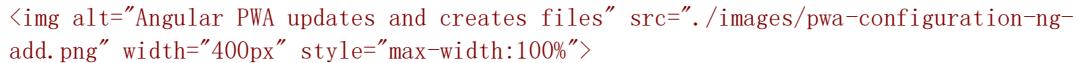
... or...

```
```shell
npm run ng add @angular/pwa
````
```

This will install the `@angular/service-worker` package and make your default app a PWA. Alternatively, you may add `project` parameter to target a specific app in your workspace:

```
```shell
yarn ng add @angular/pwa --project MyProjectName
````
```

Here is the output of the command:

A small image showing a progress bar with the text "Angular PWA updates and creates files" above it.

So, Angular CLI updates some files and add a few others:

- **ngsw-config.json** is where the [service worker configuration](https://angular.io/guide/service-worker-config) is placed. Not all PWAs have this file. It is specific to Angular.
- **manifest.webmanifest** is a [web app manifest](https://developer.mozilla.org/en-US/docs/Web/Manifest) and provides information about your app in JSON format.
- **icons** are placeholder icons that are referred to in your web app manifest. We will replace these in a minute.
- **angular.json** has following modifications:
  - `assets` include `_manifest.webmanifest_`.
  - `serviceWorker` is `true` in production build.
  - `ngswConfigPath` refers to `_ngsw-config.json_`.
- **package.json** has `_@angular/service-worker_` as a new dependency.
- **app.module.ts** imports ``ServiceWorkerModule`` and registers a service worker filename.
- **index.html** has following modifications:
  - A `<link>` element that refers to `_manifest.webmanifest_`.
  - A `<meta>` tag that sets a theme color.

## ## 2. Update the Web App Manifest

### ### 2.1. Set the Name of Your App

The `name` and the `short\_name` properties in the generated manifest are derived from your project name. Let's change them.

Open the `_manifest.webmanifest_` file and update `name` and `short\_name` props:

```
```json
{
  /* rest of the manifest meta data */
  "short_name": "My Project",
  "name": "My Project: My Catch-Phrase"
}
```

```

The short name must be really short because it will be displayed on anywhere with limited space, like the launcher and the home screen.

### ### 2.2. Add a Description

The `@angular/pwa` schematic we just added does not insert a description to your manifest file, but, according to [\[web app manifest standards\]](#) (<https://www.w3.org/TR/appmanifest/#description-member>), you should.

So, open the `_manifest.webmanifest_` file and place the description as seen below:

```
```json
{
  /* rest of the manifest meta data */
  "description": "My short project description giving a slightly better idea about my app"
}
```

```

As a bonus, providing a description [\[along with other criteria\]](#) (<https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps-edgehtml/microsoft-store#criteria-for-automatic-submission>) helps Bing web crawler to index your app and automatically submit your app to Microsoft Store in `.appx` format.

### ### 2.3. Set App Colors

Angular generates the manifest file with a default `theme_color` and `background_color`. Change these according to your brand identity:

Open the `_manifest.webmanifest_` file and update `theme_color` and `background_color` properties:

```
```json
{
  /* rest of the manifest meta data */
  "theme_color": "#000000",
  "background_color": "#ffffff"
}
```

```

Then open `_index.html_` and change the theme color meta tag as below:

```
```html
<meta name="theme-color" content="#000000" />

```

```
```
```

#### #### 2.4. Replace App Icons & Add Splash Screens

We need to update the icons and add some splash screens. This normally is time-consuming, but we will use the marvelous [pwa-asset-generator](<https://github.com/onderceylan/pwa-asset-generator#readme>) library.

First, open the `_manifest.webmanifest_` file and remove all elements in the ``icons`` property:

```
```json
{
  /* rest of the manifest meta data */
  "icons": []
}
```
```

Then, run the following command in your terminal (changing the path of course):

```
```shell
npx pwa-asset-generator /path/to/your/logo.png ./src/assets/pwa -i ./src/index.html -
m ./src/manifest.webmanifest
````
```

Open the `_manifest.webmanifest_` file again. You will see this:

```
```json
{
  /* rest of the manifest meta data */
  "icons": [
    {
      "src": "../manifest-icon-192.png",
      "sizes": "192x192",
      "type": "image/png",
      "purpose": "maskable any"
    },
    {
      "src": "../manifest-icon-512.png",
      "sizes": "512x512",
      "type": "image/png",
      "purpose": "maskable any"
    }
  ]
}
```
```

In addition to updated icons, the library will generate splash screens. However, Apple requires all splash screens to be added in your `_index.html_` and displays a blank screen at startup otherwise. So, the following tags will be inserted into the `_index.html_` file:

```
```html
```

```
<link
  rel="apple-touch-icon"
  sizes="180x180"
  href="assets/pwa/apple-icon-180.jpg"
/>
<link
  rel="apple-touch-icon"
  sizes="167x167"
  href="assets/pwa/apple-icon-167.jpg"
/>
<link
  rel="apple-touch-icon"
  sizes="152x152"
  href="assets/pwa/apple-icon-152.jpg"
/>
<link
  rel="apple-touch-icon"
  sizes="120x120"
  href="assets/pwa/apple-icon-120.jpg"
/>

<meta name="apple-mobile-web-app-capable" content="yes" />

<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2048-2732.jpg"
  media="(device-width: 1024px) and (device-height: 1366px) and (-webkit-device-pixel-
ratio: 2) and (orientation: portrait)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2732-2048.jpg"
  media="(device-width: 1024px) and (device-height: 1366px) and (-webkit-device-pixel-
ratio: 2) and (orientation: landscape)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-1668-2388.jpg"
  media="(device-width: 834px) and (device-height: 1194px) and (-webkit-device-pixel-
ratio: 2) and (orientation: portrait)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2388-1668.jpg"
  media="(device-width: 834px) and (device-height: 1194px) and (-webkit-device-pixel-
ratio: 2) and (orientation: landscape)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-1536-2048.jpg"
  media="(device-width: 768px) and (device-height: 1024px) and (-webkit-device-pixel-
ratio: 2) and (orientation: portrait)"
```

```
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2048-1536.jpg"
  media="(device-width: 768px) and (device-height: 1024px) and (-webkit-device-pixel-
ratio: 2) and (orientation: landscape)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-1668-2224.jpg"
  media="(device-width: 834px) and (device-height: 1112px) and (-webkit-device-pixel-
ratio: 2) and (orientation: portrait)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2224-1668.jpg"
  media="(device-width: 834px) and (device-height: 1112px) and (-webkit-device-pixel-
ratio: 2) and (orientation: landscape)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-1620-2160.jpg"
  media="(device-width: 810px) and (device-height: 1080px) and (-webkit-device-pixel-
ratio: 2) and (orientation: portrait)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2160-1620.jpg"
  media="(device-width: 810px) and (device-height: 1080px) and (-webkit-device-pixel-
ratio: 2) and (orientation: landscape)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-1242-2688.jpg"
  media="(device-width: 414px) and (device-height: 896px) and (-webkit-device-pixel-ratio:
3) and (orientation: portrait)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2688-1242.jpg"
  media="(device-width: 414px) and (device-height: 896px) and (-webkit-device-pixel-ratio:
3) and (orientation: landscape)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-1125-2436.jpg"
  media="(device-width: 375px) and (device-height: 812px) and (-webkit-device-pixel-ratio:
3) and (orientation: portrait)"
/>
<link
  rel="apple-touch-startup-image"
  href="assets/pwa/apple-splash-2436-1125.jpg"
```

```
    media="(device-width: 375px) and (device-height: 812px) and (-webkit-device-pixel-ratio: 3) and (orientation: landscape)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-828-1792.jpg"  
    media="(device-width: 414px) and (device-height: 896px) and (-webkit-device-pixel-ratio: 2) and (orientation: portrait)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-1792-828.jpg"  
    media="(device-width: 414px) and (device-height: 896px) and (-webkit-device-pixel-ratio: 2) and (orientation: landscape)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-1080-1920.jpg"  
    media="(device-width: 360px) and (device-height: 640px) and (-webkit-device-pixel-ratio: 3) and (orientation: portrait)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-1920-1080.jpg"  
    media="(device-width: 360px) and (device-height: 640px) and (-webkit-device-pixel-ratio: 3) and (orientation: landscape)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-750-1334.jpg"  
    media="(device-width: 375px) and (device-height: 667px) and (-webkit-device-pixel-ratio: 2) and (orientation: portrait)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-1334-750.jpg"  
    media="(device-width: 375px) and (device-height: 667px) and (-webkit-device-pixel-ratio: 2) and (orientation: landscape)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-640-1136.jpg"  
    media="(device-width: 320px) and (device-height: 568px) and (-webkit-device-pixel-ratio: 2) and (orientation: portrait)"  
/>/  
<link  
    rel="apple-touch-startup-image"  
    href="assets/pwa/apple-splash-1136-640.jpg"  
    media="(device-width: 320px) and (device-height: 568px) and (-webkit-device-pixel-ratio: 2) and (orientation: landscape)"  
/>/  
...
```

```
## 3. Configure Service Worker
```

3.1 Modify Asset Groups

Angular has defined some static files to be cached by the service worker, but they are not 100% accurate. Let's change it.

Open `_ngsw-config.json_` file and replace its content with this:

```
```json
{
 "$schema": "./node_modules/@angular/service-worker/config/schema.json",
 "index": "/index.html",
 "assetGroups": [
 {
 "name": "app",
 "installMode": "prefetch",
 "resources": {
 "files": [
 "/favicon.ico",
 "/index.html",
 "/manifest.webmanifest",
 "/*.css",
 "/common-es2015.*.js",
 "/main-es2015.*.js",
 "/polyfills-es2015.*.js",
 "/runtime-es2015.*.js",
 "/vendor-es2015.*.js"
]
 }
 },
 {
 "name": "modules",
 "installMode": "lazy",
 "updateMode": "prefetch",
 "resources": {
 "files": [
 "/*-es2015.*.js",
 "!/common-es2015.*.js",
 "!/main-es2015.*.js",
 "!/polyfills-es2015.*.js",
 "!/runtime-es2015.*.js",
 "!/vendor-es2015.*.js"
]
 }
 },
 {
 "name": "assets",
 "installMode": "lazy",
 "updateMode": "prefetch",
 "resources": {
```

```

 "files": [
 "/assets/**",
 "/*. (eot|svg|cur|jpg|png|webp|gif|otf|ttf|woff|woff2|ani)"
]
 }
]
 }
...

```

In case you want to cache other static files, please refer to the [service worker configuration document](<https://angular.io/guide/service-worker-config#assetgroups>) on Angular.io.

### ### 3.2 Set Data Groups

This part is unique to your project. We recommend being very careful about which endpoints to cache. Please refer to [service worker configuration document](<https://angular.io/guide/service-worker-config#datagroups>) on Angular.io for details.

#### 10.3.2.5 Unit Testing

##### # Unit Testing Angular UI

ABP Angular UI is tested like any other Angular application. So, [the guide here](<https://angular.io/guide/testing>) applies to ABP too. That said, we would like to point out some **\*\*unit testing topics specific to ABP Angular applications\*\***.

##### ## Setup

In Angular, unit tests use [Karma](<https://karma-runner.github.io/>) and [Jasmine](<https://jasmine.github.io>) by default. Although we like Jest more, we chose not to deviate from these defaults, so **\*\*the application template you download will have Karma and Jasmine preconfigured\*\***. You can find the Karma configuration inside the `_karma.conf.js_` file in the root folder. You don't have to do anything. Adding a spec file and running ``npm test`` will work.

##### ## Basics

An over-simplified spec file looks like this:

```

```js
import { CoreTestingModule } from "@abp/ng.core/testing";
import { ThemeBasicTestingModule } from "@abp/ng.theme.basic/testing";
import { ThemeSharedTestingModule } from "@abp/ng.theme.shared/testing";
import { ComponentFixture, TestBed, waitForAsync } from "@angular/core/testing";
import { NgxValidateCoreModule } from "@ngx-validate/core";
import { MyComponent } from "./my.component";

```

```

describe("MyComponent", () => {
  let fixture: ComponentFixture<MyComponent>;

  beforeEach(
    waitForAsync(() => {
      TestBed.configureTestingModule({
        declarations: [MyComponent],
        imports: [
          CoreTestingModule.withConfig(),
          ThemeSharedTestingModule.withConfig(),
          ThemeBasicTestingModule.withConfig(),
          NgxValidateCoreModule,
        ],
        providers: [
          /* mock providers here */
        ],
      }).compileComponents();
    })
  );

  beforeEach(() => {
    fixture = TestBed.createComponent(MyComponent);
    fixture.detectChanges();
  });

  it("should be initiated", () => {
    expect(fixture.componentInstance).toBeTruthy();
  });
});
```

```

If you take a look at the imports, you will notice that we have prepared some testing modules to replace built-in ABP modules. This is necessary for providing mocks for some features which otherwise would break your tests. Please remember to **\*\*use testing modules\*\*** and **\*\*call their `withConfig` static method\*\***.

## ## Tips

### ### Angular Testing Library

Although you can test your code with Angular TestBed, you may find [[Angular Testing Library](#)](<https://testing-library.com/docs/angular-testing-library/intro>) a good alternative.

The simple example above can be written with Angular Testing Library as follows:

```

```js
import { CoreTestingModule } from "@abp/ng.core/testing";
import { ThemeBasicTestingModule } from "@abp/ng.theme.basic/testing";
import { ThemeSharedTestingModule } from "@abp/ng.theme.shared/testing";
import { ComponentFixture } from "@angular/core/testing";
import { NgxValidateCoreModule } from "@ngx-validate/core";
```

```

```

import { render } from "@testing-library/angular";
import { MyComponent } from "./my.component";

describe("MyComponent", () => {
 let fixture: ComponentFixture<MyComponent>;

 beforeEach(async () => {
 const result = await render(MyComponent, {
 imports: [
 CoreTestingModule.withConfig(),
 ThemeSharedTestingModule.withConfig(),
 ThemeBasicTestingModule.withConfig(),
 NgxValidateCoreModule,
],
 providers: [
 /* mock providers here */
],
 });
 fixture = result.fixture;
 });

 it("should be initiated", () => {
 expect(fixture.componentInstance).toBeTruthy();
 });
});
```

```

Very similar, as you can see. The real difference kicks in when we use queries and fire events.

```

```js
// other imports
import { getByLabelText, screen } from "@testing-library/angular";
import userEvent from "@testing-library/user-event";

describe("MyComponent", () => {
 beforeEach(/* removed for sake of brevity */);

 it("should display advanced filters", () => {
 const filters = screen.getByTestId("author-filters");
 const nameInput = getByLabelText(filters, /name/i) as HTMLInputElement;
 expect(nameInput.offsetWidth).toBe(0);

 const advancedFiltersBtn = screen.getByRole("link", { name: /advanced/i });
 userEvent.click(advancedFiltersBtn);

 expect(nameInput.offsetWidth).toBeGreaterThan(0);

 userEvent.type(nameInput, "foo{backspace}");
 expect(nameInput.value).toBe("foo");
 });
});
```

```

```
});  
};
```

The **queries in Angular Testing Library follow practices for maintainable tests**, the user event package provides a **human-like interaction** with the DOM, and the library in general has **a clear API** that simplifies component testing. Please find some useful links below:

- [Queries] (<https://testing-library.com/docs/dom-testing-library/api-queries>)
- [User Event] (<https://testing-library.com/docs/ecosystem-user-event>)
- [Examples] (<https://github.com/testing-library/angular-testing-library/tree/main/apps/example-app/src/app/examples>)

Clearing DOM After Each Spec

One thing to remember is that Karma runs tests in real browser instances. That means, you will be able to see the result of your test code, but also have problems with components attached to the document body which may not get cleared after each test, even when you configure Karma to do so.

We have prepared a simple function with which you can clear any leftover DOM elements after each test.

```
```js  
// other imports
import { clearPage } from "@abp/ng.core/testing";

describe("MyComponent", () => {
 let fixture: ComponentFixture<MyComponent>;

 afterEach(() => clearPage(fixture));

 beforeEach(async () => {
 const result = await render(MyComponent, {
 /* removed for sake of brevity */
 });
 fixture = result.fixture;
 });

 // specs here
});
```
```

Please make sure you use it because Karma will fail to remove dialogs otherwise and you will have multiple copies of modals, confirmation boxes, and alike.

Waiting

Some components, modals, in particular, work off-detection-cycle. In other words, you cannot reach DOM elements inserted by these components immediately after opening them. Similarly, inserted elements are not immediately destroyed upon closing them.

For this purpose, we have prepared a `wait` function.

```
```js
// other imports
import { wait } from "@abp/ng.core/testing";

describe("MyComponent", () => {
 beforeEach(/* removed for sake of brevity */);

 it("should open a modal", async () => {
 const openModalBtn = screen.getByRole("button", { name: "Open Modal" });
 userEvent.click(openModalBtn);

 await wait(fixture);

 const modal = screen.getByRole("dialog");
 expect(modal).toBeTruthy();

 /* wait again after closing the modal */
 });
});
```

```

The `wait` function takes a second parameter, i.e. timeout (default: `0`). Try not to use it though. Using a timeout bigger than `0` is usually a signal that something is not quite right.

Testing Example

Here is an example test suite. It doesn't cover all, but gives quite a good idea about what the testing experience will be like.

```
```js
import { clearPage, CoreTestingModule, wait } from "@abp/ng.core/testing";
import { ThemeBasicTestingModule } from "@abp/ng.theme.basic/testing";
import { ThemeSharedTestingModule } from "@abp/ng.theme.shared/testing";
import { ComponentFixture } from "@angular/core/testing";
import {
 NgbCollapseModule,
 NgbDatepickerModule,
 NgbDropdownModule,
} from "@ng-bootstrap/ng-bootstrap";
import { NgxValidateCoreModule } from "@ngx-validate/core";
import { CountryService } from "@proxy/countries";
import {
 findByText,
 getByLabelText,
 getByRole,
 getText,
 queryByRole,
 render,
}
```

```

 screen,
} from "@testing-library/angular";
import userEvent from "@testing-library/user-event";
import { BehaviorSubject, of } from "rxjs";
import { CountryComponent } from "./country.component";

const list$ = new BehaviorSubject({
 items: [{ id: "ID_US", name: "United States of America" }],
 totalCount: 1,
});

describe("Country", () => {
 let fixture: ComponentFixture<CountryComponent>;

 afterEach(() => clearPage(fixture));

 beforeEach(async () => {
 const result = await render(CountryComponent, {
 imports: [
 CoreTestingModule.withConfig(),
 ThemeSharedTestingModule.withConfig(),
 ThemeBasicTestingModule.withConfig(),
 NgxValidateCoreModule,
 NgbCollapseModule,
 NgbDatepickerModule,
 NgbDropdownModule,
],
 providers: [
 {
 provide: CountryService,
 useValue: {
 getList: () => list$,
 },
 },
],
 });
 fixture = result.fixture;
 });

 it("should display advanced filters", () => {
 const filters = screen.getByTestId("country-filters");
 const nameInput = getLabelText(filters, /name/i) as HTMLInputElement;
 expect(nameInput.offsetWidth).toBe(0);

 const advancedFiltersBtn = screen.getByRole("link", { name: /advanced/i });
 userEvent.click(advancedFiltersBtn);

 expect(nameInput.offsetWidth).toBeGreaterThan(0);

 userEvent.type(nameInput, "foo{backspace}");
 expect(nameInput.value).toBe("foo");
 });
});

```

```
 userEvent.click(advancedFiltersBtn);
 expect(nameInput.offsetWidth).toBe(0);
 });

it("should have a heading", () => {
 const heading = screen.getByRole("heading", { name: "Countries" });
 expect(heading).toBeInTheDocument();
});

it("should render list in table", async () => {
 const table = await screen.findByTestId("country-table");

 const name = getByText(table, "United States of America");
 expect(name).toBeInTheDocument();
});

it("should display edit modal", async () => {
 const actionsBtn = screen.queryByRole("button", { name: /actions/i });
 userEvent.click(actionsBtn);

 const editBtn = screen.getByRole("button", { name: /edit/i });
 userEvent.click(editBtn);

 await wait(fixture);

 const modal = screen.getByRole("dialog");
 const modalHeading = queryByRole(modal, "heading", { name: /edit/i });
 expect(modalHeading).toBeInTheDocument();

 const closeBtn = getByText(modal, "×");
 userEvent.click(closeBtn);

 await wait(fixture);

 expect(screen.queryByRole("dialog")).toBeFalsy();
});

it("should display create modal", async () => {
 const newBtn = screen.getByRole("button", { name: /new/i });
 userEvent.click(newBtn);

 await wait(fixture);

 const modal = screen.getByRole("dialog");
 const modalHeading = queryByRole(modal, "heading", { name: /new/i });

 expect(modalHeading).toBeInTheDocument();
});

it("should validate required name field", async () => {
 const newBtn = screen.getByRole("button", { name: /new/i });

```

```

 userEvent.click(newBtn);

 await wait(fixture);

 const modal = screen.getByRole("dialog");
 const nameInput = getByRole(modal, "textbox", {
 name: /^name/i,
 }) as HTMLInputElement;

 userEvent.type(nameInput, "x");
 userEvent.type(nameInput, "{backspace}");

 const nameError = await findByText(modal, /required/i);
 expect(nameError).toBeTruthy();
 });

 it("should delete a country", () => {
 const getSpy = spyOn(fixture.componentInstance.list, "get");
 const deleteSpy = jasmine.createSpy().and.returnValue(of(null));
 fixture.componentInstance.service.delete = deleteSpy;

 const actionsBtn = screen.queryByRole("button", { name: /actions/i });
 userEvent.click(actionsBtn);

 const deleteBtn = screen.getByRole("button", { name: /delete/i });
 userEvent.click(deleteBtn);

 const confirmText = screen.getText("AreYouSure");
 expect(confirmText).toBeTruthy();

 const confirmBtn = screen.getByRole("button", { name: "Yes" });
 userEvent.click(confirmBtn);

 expect(deleteSpy).toHaveBeenCalledWith(list$.value.items[0].id);
 expect(getSpy).toHaveBeenCalledTimes(1);
 });
});
```

```

CI Configuration

You would need a different configuration for your CI environment. To set up a new configuration for your unit tests, find the test project in `_angular.json` file and add one as seen below:

```

```json
// angular.json

"test": {
 "builder": "@angular-devkit/build-angular:karma",
 "options": { /* several options here */ },
 "configurations": {

```

```
 "production": {
 "karmaConfig": "karma.conf.prod.js"
 }
 }
}
```

```

Now you can copy the `_karma.conf.js_` as `_karma.conf.prod.js_` and use any configuration you like in it. Please check [[Karma configuration file document](#)] (<http://karma-runner.github.io/5.2/config/configuration-file.html>) for config options.

Finally, don't forget to run your CI tests with the following command:

```
```sh
npm test -- --prod
```

```

See Also

- [[ABP Community Video – Unit Testing with the Angular UI](#)] (<https://community.abp.io/articles/unit-testing-with-the-angular-ui-p41550q3>)

10.3.3 Core Functionality

10.3.3.1 Config State Service

Config State Service

``ConfigStateService`` is a singleton service, i.e. provided in root level of your application, and keeps the application configuration response in the internal store.

Before Use

In order to use the ``ConfigStateService`` you must inject it in your class as a dependency.

```
```js
import { ConfigStateService } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 constructor(private config: ConfigStateService) {}
}
```

```

You do not have to provide the ``ConfigStateService`` at module or component/directive level, because it is already ****provided in root****.

Get Methods

``ConfigStateService`` has numerous get methods which allow you to get a specific configuration or all configurations.

Get methods with `"$"` at the end of the method name (e.g. ``getAll$``) return an RxJs stream. The streams are triggered when set or patched the state.

How to Get All Configurations

You can use the ``getAll`` or ``getAll$`` method of ``ConfigStateService`` to get all of the application configuration response object. It is used as follows:

```
```js
// this.config is instance of ConfigStateService

const config = this.config.getAll();

// or
this.config.getAll$.subscribe(config => {
 // use config here
})
```

```

How to Get a Specific Configuration

You can use the ``getOne`` or ``getOne$`` method of ``ConfigStateService`` to get a specific configuration property. For that, the property name should be passed to the method as parameter.

```
```js
// this.config is instance of ConfigStateService

const currentUser = this.config.getOne("currentUser");

// or
this.config.getOne$("currentUser").subscribe(currentUser => {
 // use currentUser here
})
```

```

On occasion, you will probably want to be more specific than getting just the current user. For example, here is how you can get the ``tenantId``:

```
```js
const tenantId = this.config.getDeep("currentUser.tenantId");

// or
this.config.getDeep$("currentUser.tenantId").subscribe(tenantId => {
 // use tenantId here
})
```

```

or by giving an array of keys as parameter:

```
```js
const tenantId = this.config.getDeep(["currentUser", "tenantId"]);
```
```

FYI, `getDeep` is able to do everything `getOne` does. Just keep in mind that `getOne` is slightly faster.

How to Get a Feature

You can use the `getFeature` or `getFeature\$` method of `ConfigStateService` to get a feature value. For that, the feature name should be passed to the method as parameter.

```
```js
// this.config is instance of ConfigStateService

const enableLdapLogin = this.config.getFeature("Account.EnableLdapLogin");

// or
this.config.getFeature$("Account.EnableLdapLogin").subscribe(enableLdapLogin => {
 // use enableLdapLogin here
})
```
```

› For more information, see the [[features document](#)] (./Features).

How to Get a Setting

You can use the `getSetting` or `getSetting\$` method of `ConfigStateService` to get a setting. For that, the setting name should be passed to the method as parameter.

```
```js
// this.config is instance of ConfigStateService

const twoFactorBehaviour = this.config.getSetting("Abp.Identity.TwoFactor.Behaviour");

// or
this.config.getSetting$("Abp.Identity.TwoFactor.Behaviour").subscribe(twoFactorBehaviour
=> {
 // use twoFactorBehaviour here
})
```
```

› For more information, see the [[settings document](#)] (./Settings).

State Properties

Please refer to `ApplicationConfigurationDto` type for all the properties you can get with `getOne` and `getDeep`. It can be found in the [[models.ts file](#)] (<https://github.com/abpframework/abp/blob/dev/npm/ng-packs/packages/core/src/lib/proxy/volo/abp/asp-net-core/mvc/application-configurations/models.ts#L11>).

Set State

`ConfigStateService` has a method named `setState` which allow you to set the state value.

You can get the application configuration response and set the `ConfigStateService` state value as shown below:

```
```js
import {AbpApplicationConfigurationService, ConfigStateService} from '@abp/ng.core';

constructor(private abpApplicationConfigurationService: AbpApplicationConfigurationService, private config: ConfigStateService) {
 this.abpApplicationConfigurationService.get().subscribe(config => {
 this.config.setState(config);
 })
}
```

```

See Also

- [Settings] (./Settings.md)
- [Features] (./Features.md)

10.3.3.2 Authorization

Authorization in Angular UI

OAuth is preconfigured in Angular application templates. So, when you start a project using the CLI (or Suite, for that matter), authorization already works. ABP Angular UI packages are using [angular-oauth2-oidc library](<https://github.com/manfredsteyer/angular-oauth2-oidc#logging-in>) for managing OAuth in the Angular client.

You can find ****OAuth configuration**** in the `_environment.ts` files.

Authorization Code Flow

```
```js
import { Config } from '@abp/ng.core';

const baseUrl = 'http://localhost:4200';

export const environment = {
 // other options removed for sake of brevity

 oAuthConfig: {
 issuer: 'https://localhost:44305',
 redirectUri: baseUrl,
 clientId: 'MyProjectName_App',
 responseType: 'code',
 }
}
```

```

```

    scope: 'offline_access MyProjectName',
  },
  // other options removed for sake of brevity
} as Config.Environment;
```

```

This configuration results in an [OAuth authorization code flow with PKCE] (<https://tools.ietf.org/html/rfc7636>).

According to this flow, the user is redirected to an external login page which is built with MVC. So, if you need \*\*to customize the login page\*\*, please follow [[this community article](#)] (<https://community.abp.io/articles/how-to-customize-the-login-page-for-mvc-razor-page-applications-9a40f3cd>).

#### #### Resource Owner Password Flow

If you have used the [[Angular UI account module](#)] ([./Account-Module](#)) in your project, you can switch to the resource owner password flow by changing the OAuth configuration in the `_environment.ts_` files as shown below:

```

```js
import { Config } from '@abp/ng.core';

export const environment = {
  // other options removed for sake of brevity

  oAuthConfig: {
    issuer: 'https://localhost:44305',
    clientId: 'MyProjectName_App',
    dummyClientSecret: '1q2w3e*',
    scope: 'offline_access MyProjectName',
  },
  // other options removed for sake of brevity
} as Config.Environment;
```

```

According to this flow, the user is redirected to the login page in the account module.

#### [10.3.3.3 Current User](#)

##### # Angular UI: Current User

The current user information stored in Config State.

#### #### How to Get a Current User Information Configuration

You can use the `getOne` or `getOne\$` method of `ConfigStateService` to get a specific configuration property. For that, the property name should be passed to the method as parameter.

```
```js
// this.config is an instance of ConfigStateService

const currentUser = this.config.getOne("currentUser");

// or
this.config.getOne$("currentUser").subscribe(currentUser => {
    // use currentUser here
})
```

```

➤ See the [ConfigStateService](./Config-State-Service) for more information.

#### 10.3.3.4 HTTP Requests

# How to Make HTTP Requests

##### ## About HttpClient

Angular has the amazing [HttpClient](https://angular.io/guide/http) for communication with backend services. It is a layer on top and a simplified representation of [XMLHttpRequest Web API](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest). It also is the recommended agent by Angular for any HTTP request. There is nothing wrong with using the `HttpClient` in your ABP project.

However, `HttpClient` leaves error handling to the caller (method). In other words, HTTP errors are handled manually and by hooking into the observer of the `Observable` returned.

```
```js
getConfig() {
    this.http.get(this.configUrl).subscribe(
        config => this.updateConfig(config),
        error => {
            // Handle error here
        },
    );
}
```

```

Although clear and flexible, handling errors this way is repetitive work, even when error processing is delegated to the store or any other injectable.

An `HttpInterceptor` is able to catch `HttpErrorResponse` and can be used for a centralized error handling. Nevertheless, cases where default error handler, therefore the

interceptor, must be disabled require additional work and comprehension of Angular internals. Check [[this issue](#)] (<https://github.com/angular/angular/issues/20203>) for details.

## ## RestService

ABP core module has a utility service for HTTP requests: `RestService`. Unless explicitly configured otherwise, it catches HTTP errors and dispatches a `RestOccurError` action. This action is then captured by the `ErrorHandler` introduced by the `ThemeSharedModule`. Since you should already import this module in your app, when the `RestService` is used, all HTTP errors get automatically handled by default.

### #### Getting Started with RestService

In order to use the `RestService`, you must inject it in your class as a dependency.

```
```js
import { RestService } from '@abp/ng.core';

@Injectable({
  /* class metadata here */
})
class DemoService {
  constructor(private rest: RestService) {}
}
```

```

You do not have to provide the `RestService` at module or component/directive level, because it is already **provided in root\*\***.

### #### How to Make a Request with RestService

You can use the `request` method of the `RestService` is for HTTP requests. Here is an example:

```
```js
getFoo(id: number) {
  const request: Rest.Request<null> = {
    method: 'GET',
    url: '/api/some/path/to/foo/' + id,
  };

  return this.rest.request<null, FooResponse>(request);
}
```

```

The `request` method always returns an `Observable<T>`. Therefore you can do the following wherever you use `getFoo` method:

```
```js
doSomethingWithFoo(id: number) {
  this.demoService.getFoo(id).subscribe(
    foo => {
      // Do something with foo.
    }
)
```
```

```

****You do not have to worry about unsubscription.**** The `RestService` uses `HttpClient` behind the scenes, so every observable it returns is a finite observable, i.e. it closes subscriptions automatically upon success or error.

As you see, `request` method gets a request options object with `Rest.Request<T>` type. This generic type expects the interface of the request body. You may pass `null` when there is no body, like in a `GET` or a `DELETE` request. Here is an example where there is one:

```
```js
postFoo(body: Foo) {
 const request: Rest.Request<Foo> = {
 method: 'POST',
 url: '/api/some/path/to/foo',
 body
 };

 return this.rest.request<Foo, FooResponse>(request);
}
```
```

```

You may [check here](<https://github.com/abpframework/abp/blob/dev/npm/ng-packs/packages/packages/core/src/lib/models/rest.ts#L23>) for complete `Rest.Request<T>` type, which has only a few changes compared to [HttpRequest](<https://angular.io/api/common/http/HttpRequest>) class in Angular.

#### How to Disable Default Error Handler of RestService

The ``request`` method, used with defaults, always handles errors. Let's see how you can change that behavior and handle errors yourself:

```
```js
deleteFoo(id: number) {
  const request: Rest.Request<null> = {
    method: 'DELETE',
    url: '/api/some/path/to/foo/' + id,
  };

  return this.rest.request<null, void>(request, { skipHandleError: true });
}
```

```

``skipHandleError`` config option, when set to ``true``, disables the error handler and the returned observable starts throwing an error that you can catch in your subscription.

```
```js
removeFooFromList(id: number) {
  this.demoService.deleteFoo(id).subscribe(
    foo => {
      // Do something with foo.
    },
    error => {
      // Do something with error.
    }
)
```

```

#### #### How to Get a Specific API Endpoint From Application Config

Another nice config option that ``request`` method receives is ``apiName`` (available as of v2.4), which can be used to get a specific module endpoint from application configuration.

```
```js
putFoo(body: Foo, id: string) {
  const request: Rest.Request<Foo> = {
    method: 'PUT',
    url: '/' + id,
    body
  };

  return this.rest.request<Foo, void>(request, {apiName: 'foo'});
}
```

```

`putFoo` above will request `https://localhost:44305/api/some/path/to/foo/{id}` as long as the environment variables are as follows:

```
```js
// environment.ts

export const environment = {
  apis: {
    default: {
      url: 'https://localhost:44305',
    },
    foo: {
      url: 'https://localhost:44305/api/some/path/to/foo',
    },
  },
  /* rest of the environment variables here */
}
```
```

```

How to Observe Response Object or HTTP Events Instead of Body

`RestService` assumes you are generally interested in the body of a response and, by default, sets `observe` property as `body`. However, there may be times you are rather interested in something else, such as a custom proprietary header. For that, the `request` method receives `observe` property in its config object.

```
```js
getSomeCustomHeaderValue() {
 const request: Rest.Request<null> = {
 method: 'GET',
 url: '/api/some/path/that/sends/some-custom-header',
 };

 return this.rest.request<null, HttpResponse<any>>(
 request,
 {observe: Rest.Observe.Response},
).pipe(
 map(response => response.headers.get('Some-Custom-Header'))
);
}
```
```

```

You may find `Rest.Observe` enum [here](<https://github.com/abpframework/abp/blob/dev/npm/ng-packs/packages/core/src/lib/models/rest.ts#L10>).

## ## HTTP Error Handling

When the ``RestService`` is used, all HTTP errors are reported to the [ `HttpErrorReporterService` ](./HTTP-Error-Reporter-Service), and then `ErrorHandler`, a service exposed by the `@abp/ng. theme. shared` package automatically handles the errors.

### ### Custom HTTP Error Handler

A custom HTTP error handler can be registered to an injection token named `HTTP\_ERROR\_HANDLER`. If a custom handler function is registered, the `ErrorHandler` executes that function.

See an example:

```
```js
// http-error-handler.ts
import { ContentProjectionService, PROJECTION_STRATEGY } from '@abp/ng.core';
import { ToasterService } from '@abp/ng.theme.shared';
import { HttpErrorResponse } from '@angular/common/http';
import { Injector } from '@angular/core';
import { throwError } from 'rxjs';
import { Error404Component } from './error404/error404.component';

export function handleHttpErrors(injector: Injector, httpError: HttpErrorResponse) {
    if (httpError.status === 400) {
        const toaster = injector.get(ToasterService);
        toaster.error(httpError.error?.error?.message || 'Bad request!', '400');
        return;
    }

    if (httpError.status === 404) {
        const contentProjection = injector.get(ContentProjectionService);

        contentProjection.projectContent(PROJECTION_STRATEGY, AppendComponentToBody(Error404Component));
        return;
    }

    return throwError(httpError);
}

// app.module.ts
import { Error404Component } from './error404/error404.component';
import { handleHttpErrors } from './http-error-handling';
import { HTTP_ERROR_HANDLER, ... } from '@abp/ng.theme.shared';

@NgModule({
    // ...
    providers: [
        // ...
        { provide: HTTP_ERROR_HANDLER, useValue: handleHttpErrors }
    ],
})
```

```

declarations: [
  //...
  Error404Component],
})
export class AppModule {}  

```

```

In the example above:

- Created a function named `handleHttpErrors` and defined as value of the `HTTP\_ERROR\_HANDLER` provider in app.module. After this, the function executes when an HTTP error occurs.
- 400 bad request errors is handled. When a 400 error occurs, backend error message will be displayed as shown below:

![custom-error-handler-toaster-message](images/custom-error-handler-toaster-message.jpg)

- 404 not found errors is handled. When a 404 error occurs, `Error404Component` will be appended to the `` as shown below:

![custom-error-handler-404-component](images/custom-error-handler-404-component.jpg)

- Since `throwError(httpError)` is returned at bottom of the `handleHttpErrors`, the `ErrorHandler` will handle the HTTP errors except 400 and 404 errors.

**\*\*Note 1:\*\*** If you put `return` to next line of handling an error, default error handling will not work for that error.

```

```js
export function handleHttpErrors(injector: Injector, httpError: HttpErrorResponse) {
  if (httpError.status === 403) {
    // handle 403 errors here
    return; // put return to skip default error handling
  }
}
```

```

**\*\*Note 2:\*\*** If you put `return throwError(httpError)`, default error handling will work.

- `throwError` is a function. It can be imported from `rxjs`.
- `httpError` is the second parameter of the error handler function which is registered to the `HTTP\_ERROR\_HANDLER` provider. Type of the `httpError` is `HttpErrorResponse`.

```

```js
import { throwError } from 'rxjs';

export function handleHttpErrors(injector: Injector, httpError: HttpErrorResponse) {
  if (httpError.status === 500) {
    // handle 500 errors here
    return;
  }
}
```

```

```

 // you can return the throwError(httpError) at bottom of the function to run the default
 // handler of ABP for HTTP errors that you didn't handle above.
 return throwError(httpError)
}
```

```

How to Skip HTTP interceptors and ABP headers

The ABP Framework adds several HTTP headers to the HttpClient, such as the "Auth token" or "tenant Id".

The ABP Server must possess the information but the ABP user may not want to send this informations to an external server.

ExternalHttpClient and IS EXTERNAL REQUEST HttpContext Token were added in V6.0.4.

The ABP Http interceptors check the value of the `IS_EXTERNAL_REQUEST` token. If the token is True then ABP-specific headers won't be added to Http Request.

The `ExternalHttpClient` extends from `HTTPClient` and sets the `IS_EXTERNAL_REQUEST` context token to true.

When you are using `ExternalHttpClient` as HttpClient in your components, it does not add ABP-specific headers.

Note: With `IS_EXTERNAL_REQUEST` or without it, ABP loading service works.

10.3.3.5 Localization

Localization

Before you read about *_the Localization Pipe_* and *_the Localization Service_*, you should know about localization keys.

The Localization key format consists of 2 sections which are ****Resource Name**** and ****Key****.

`ResourceName::Key`

> If you do not specify the resource name, it will be `default resourceName` which is declared in `environment.ts`

```

```js
const environment = {
 ...
 localization: {
 defaultResourceName: 'MyProjectName',
 },
};
```

```

So these two are the same:

```

```html
<h1>{${{::Key' | abpLocalization}}%}</h1>

```

```
```html
<h1>{{{{ 'MyProjectName::Key' | abpLocalization }}}%}</h1>
````
```

### ## Using the Localization Pipe

You can use the `abpLocalization` pipe to get localized text as in this example:

```
```html
<h1>{{{{ 'Resource::Key' | abpLocalization }}}%}</h1>
````
```

The pipe will replace the key with the localized text.

You can also specify a default value as shown below:

```
```html
<h1>
  {{{ { key: 'Resource::Key', defaultValue: 'Default Value' } | abpLocalization }}}%
</h1>
````
```

To use interpolation, you must give the values for interpolation as pipe parameters, for example:

Localization data is stored in key-value pairs:

```
```js
{
  //...
  AbpAccount: { // AbpAccount is the resource name
    Key: "Value",
    PagerInfo: "Showing {0} to {1} of {2} entries"
  }
}
````
```

So we can use this key like this:

```
```html
<h1>{{{ 'AbpAccount::PagerInfo' | abpLocalization:'20':'30':'50' }}}%</h1>
<!-- Output: Showing 20 to 30 of 50 entries -->
````
```

### ### Using the Localization Service

First of all you should import the `LocalizationService` from **\*\*@abp/ng.core\*\***

```
```js
import { LocalizationService } from '@abp/ng.core';
````
```

```
class MyClass {
 constructor(private localizationService: LocalizationService) {}
}
```

```

After that, you are able to use localization service.

› You can add interpolation parameters as arguments to `instant()` and `get()` methods.

```
```js
this.localizationService.instant(
 'AbpIdentity::UserDeletionConfirmation',
 'John'
);

// with fallback value
this.localizationService.instant(
 {
 key: 'AbpIdentity::UserDeletionConfirmation',
 defaultValue: 'Default Value',
 },
 'John'
);

// Output
// User 'John' will be deleted. Do you confirm that?
```

```

To get a localized text as [*Observable*] (<https://rxjs.dev/guide/observable>) use `get` method instead of `instant`:

```
```js
this.localizationService.get('Resource::Key');

// with fallback value
this.localizationService.get({
 key: 'Resource::Key',
 defaultValue: 'Default Value',
});
```

```

UI Localizations

Localizations can be determined on backend side. Angular UI gets the localizations from the `application-configuration` API's response. You can also determine localizations on the UI side.

See an example:

```
```ts
// app.module.ts

```

```

@NgModule({
 imports: [
 //...other imports
 CoreModule.forRoot({
 localizations: [
 {
 culture: 'en',
 resources: [
 {
 resourceName: 'MyProjectName',
 texts: {
 Administration: 'Administration',
 HomePage: 'Home',
 },
 },
],
 },
 {
 culture: 'de',
 resources: [
 {
 resourceName: 'MyProjectName',
 texts: {
 Administration: 'Verwaltung',
 HomePage: 'Startseite',
 },
 },
],
 },
],
 })),
]
})
```

```

...or, you can determine the localizations in a feature module:

```

```ts
// your feature module

@NgModule({
 imports: [
 //...other imports
 CoreModule.forChild({
 localizations: [
 {
 culture: 'en',
 resources: [
 {
 resourceName: 'MyProjectName',
 texts: {
 Administration: 'Administration',
 },
 },
],
 },
],
 })),
]
})
```

```

```

        HomePage: 'Home',
    },
},
],
},
{
culture: 'de-DE',
resources: [
{
resourceName: 'MyProjectName',
texts: {
Administration: 'Verwaltung',
HomePage: 'Startseite',
},
},
],
},
],
}),
]
})
```

```

The localizations above can be used like this:

```

```html
<div>{{{'MyProjectName':Administration} | abpLocalization}}</div>

<div>{{{'MyProjectName':HomePage} | abpLocalization}}</div>
```

```

> **\*\*Note:\*\*** If you have specified the same localizations in the UI and backend, the backend localizations override the UI localizations.

## ## RTL Support

As of v2.9 ABP has RTL support. If you are generating a new project with v2.9 and above, everything is set, you do not need to do any changes. If you are migrating your project from an earlier version, please follow the 2 steps below:

### #### Step 1. Create Chunks for Bootstrap LTR and RTL

Find [styles configuration in angular.json] (<https://angular.io/guide/workspace-config#style-script-config>) and make sure the chunks in your project has `bootstrap-rtl.min` and `bootstrap-ltr.min` as shown below.

```

```json
{
"projects": {
"MyProjectName": {
"architect": {
"build": {

```

```

"options": {
  "styles": [
    {
      "input": "node_modules/@fortawesome/fontawesome-free/css/all.min.css",
      "inject": true,
      "bundleName": "fontawesome-all.min"
    },
    {
      "input": "node_modules/@fortawesome/fontawesome-free/css/v4-
shims.min.css",
      "inject": true,
      "bundleName": "fontawesome-v4-shims.min"
    },
    {
      "input": "node_modules/@abp/ng.theme.shared/styles/bootstrap-rtl.min.css",
      "inject": false,
      "bundleName": "bootstrap-rtl.min"
    },
    {
      "input": "node_modules/bootstrap/dist/css/bootstrap.min.css",
      "inject": true,
      "bundleName": "bootstrap-ltr.min"
    },
    "apps/dev-app/src/styles.scss"
  ]
}
}
}
}
```

```

#### #### Step 2. Clear Lazy Loaded Fontawesome in AppComponent

If you have created and injected chunks for Fontawesome as seen above, you no longer need the lazy loading in the `AppComponent` which was implemented before v2.9. Simply remove them. The `AppComponent` in the template of the new version looks like this:

```

```js
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <abp-loader-bar></abp-loader-bar>
    <router-outlet></router-outlet>
  `,
})
export class AppComponent {}
```

```

## ## Registering a New Locale

Since ABP has more than one language, Angular locale files loads lazily using [Webpack's import function](<https://webpack.js.org/api/module-methods/#import-1>) to avoid increasing the bundle size and register to Angular core using the [`registerLocaleData`](<https://angular.io/api/common/registerLocaleData>) function. The chunks to be included in the bundle are specified by the [Webpack's magic comments](<https://webpack.js.org/api/module-methods/#magic-comments>) as hard-coded. Therefore a `registerLocale` function that returns Webpack `import` function must be passed to `CoreModule`.

### ### registerLocaleFn

`registerLocale` function that exported from `@abp/ng.core/locale` package is a higher order function that accepts `cultureNameLocaleFileMap` object and `errorHandlerFn` function as params and returns Webpack `import` function. A `registerLocale` function must be passed to the `forRoot` of the `CoreModule` as shown below:

```
```js
// app.module.ts

import { registerLocale } from '@abp/ng.core/locale';
// if you have commercial license and the language management module, add the below import
// import { registerLocale } from '@volo/abp.ng.language-management/locale';

@NgModule({
  imports: [
    // ...
    CoreModule.forRoot({
      // ...other options,
      registerLocaleFn: registerLocale(
        // you can pass the cultureNameLocaleFileMap and errorHandlerFn as optionally
        {
          cultureNameLocaleFileMap: { 'pt-BR': 'pt' },
          errorHandlerFn: ({ resolve, reject, locale, error }) => {
            // the error can be handled here
          },
        },
        )
      ),
      //...
    ])
  }
})
```

Mapping of Culture Name to Angular Locale File Name

Some of the culture names defined in .NET do not match Angular locales. In such cases, the Angular app throws an error like below at runtime:

```
![locale-error] ./images/locale-error.png
```

If you see an error like this, you should pass the `cultureNameLocaleFileMap` property like below to the `registerLocale` function.

```
```js
// app.module.ts

import { registerLocale } from '@abp/ng.core/locale';
// if you have commercial license and the language management module, add the below import
// import { registerLocale } from '@volo/abp.ng.language-management/locale';

@NgModule({
 imports: [
 // ...
 CoreModule.forRoot({
 // ...other options,
 registerLocaleFn: registerLocale(
 {
 cultureNameLocaleFileMap: {
 "DotnetCultureName": "AngularLocaleFileName",
 "pt-BR": "pt" // example
 },
 },
)
)),
 //...
]
})
```

See [all locale files in Angular](<https://github.com/angular/angular/tree/master/packages/common/locales>).

#### #### Adding a New Culture

Add the below code to the `app.module.ts` by replacing `your-locale` placeholder with a correct locale name.

```
```js
//app.module.ts

import { storeLocaleData } from '@abp/ng.core/locale';
import(
  /* webpackChunkName: "_locale-your-locale-js"*/
  /* webpackMode: "eager" */
  '@angular/common/locales/your-locale.js'
).then((m) => storeLocaleData(m.default, 'your-locale'));

```
```

...or a custom `registerLocale` function can be passed to the `CoreModule`:

```
```js
// register-locale.ts
```

```

import { differentLocales } from '@abp/ng.core';
export function registerLocale(locale: string) {
    return import(
        /* webpackChunkName: '_locale-[request]"*/
        /* webpackInclude: /[\\](en|fr).js/ */
        /* webpackExclude: /[\\]global|extra/ */
        `@angular/common/locales/${differentLocales[locale] || locale}.js`
    )
}

// app.module.ts

import { registerLocale } from './register-locale';

@NgModule({
    imports: [
        // ...
        CoreModule.forRoot({
            // ...other options,
            registerLocaleFn: registerLocale
        }),
        //...
    ]
})
```

```

After this custom `registerLocale` function, since the en and fr added to the `webpackInclude`, only en and fr locale files will be created as chunks:

![locale chunks](https://user-images.githubusercontent.com/34455572/98203212-acaa2100-1f44-11eb-85af-4eb66d296326.png)

Which locale files you add to `webpackInclude` magic comment, they will be included in the bundle

#### ## See Also

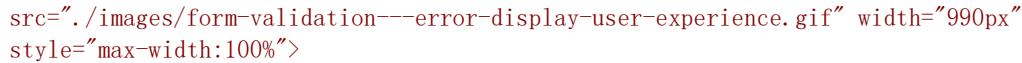
- [Localization in ASP.NET Core](../../Localization.md)

#### *10.3.3.6 Form Validation*

##### # Form Validation

Reactive forms in ABP Angular UI are validated by [ngx-validate](https://www.npmjs.com/package/@ngx-validate/core) and helper texts are shown automatically based on validation rules and error blueprints. You do not have to add any elements or components to your templates. The library handles that for you. Here is how the experience is:

<img alt="The ngx-validate library validates an Angular reactive form and an error text appears under each wrong input based on the validation rule and the error blueprint.">

A screenshot of a user interface showing a form with several input fields. Some fields have red borders around them, indicating they are invalid or have errors. Error messages are displayed below each of these fields in a small, red font.

## ## How to Add New Error Messages

You can add a new error message by passing validation options to the `ThemeSharedModule` in your root module.

```
```js
import { VALIDATION_BLUEPRINTS } from "@ngx-validate/core";
import { DEFAULT_VALIDATION_BLUEPRINTS } from "@abp/ng.theme.shared";

@NgModule({
  imports: [
    ThemeSharedModule.forRoot({
      validation: {
        blueprints: {
          uniqueUsername: "::AlreadyExists[%{{{ username }}}%]",

        },
      },
      // rest of theme shared config
    }),
    // other imports
  ],
  // rest of the module metadata
})
export class AppModule {}
````
```

Alternatively, you may provide the `VALIDATION\_BLUEPRINTS` token directly in your root module. Please do not forget to spread `DEFAULT\_VALIDATION\_BLUEPRINTS`. Otherwise, built-in ABP validation messages will not work.

```
```js
import { VALIDATION_BLUEPRINTS } from "@ngx-validate/core";
import { DEFAULT_VALIDATION_BLUEPRINTS } from "@abp/ng.theme.shared";

@NgModule({
  providers: [
    {
      provide: VALIDATION_BLUEPRINTS,
      useValue: {
        ...DEFAULT_VALIDATION_BLUEPRINTS,
        uniqueUsername: "::AlreadyExists[%{{{ username }}}%]",

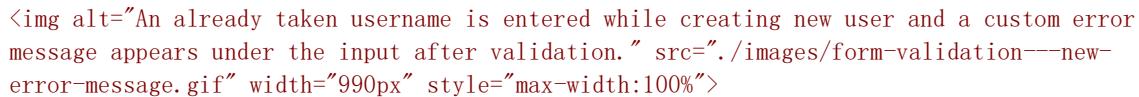
      },
    },
    // other providers
  ],
````
```

```

 // rest of the module metadata
})
export class AppModule {}
```

```

When a [validator](<https://angular.io/guide/form-validation#defining-customValidators>) or an [async validator](<https://angular.io/guide/form-validation#creating-asynchronousValidators>) returns an error with the key given to the error blueprints (`uniqueUsername` here), the validation library will be able to display an error message after localizing according to the given key and interpolation params. The result will look like this:



An already taken username is entered while creating new user and a custom error message appears under the input after validation.

In this example;

- Localization key is `::AlreadyExists`.
- The interpolation param is `username`.
- Localization resource is defined as `":AlreadyExists": "Sorry, '{0}' already exists."`.
- And the validator should return `{ uniqueUsername: { username: "admin" } }` as the error object.

How to Change Existing Error Messages

You can overwrite an existing error message by passing validation options to the `ThemeSharedModule` in your root module. Let's imagine you have a custom localization resource for required inputs.

```

```json
"RequiredInput": "Oops! We need this input."
```

```

To use this instead of the built-in required input message, all you need to do is the following.

```

```js
import { VALIDATION_BLUEPRINTS } from "@ngx-validate/core";
import { DEFAULT_VALIDATION_BLUEPRINTS } from "@abp/ng.theme.shared";

@NgModule({
 imports: [
 ThemeSharedModule.forRoot({
 validation: {
 blueprints: {
 required: "::RequiredInput",
 },
 },
 }),
 // rest of theme shared config
}),

```

```

 // other imports
],
 // rest of the module metadata
})
export class AppModule {}
```

```

Alternatively, you may provide the `VALIDATION_BLUEPRINTS` token directly in your root module. Please do not forget to spread `DEFAULT_VALIDATION_BLUEPRINTS`. Otherwise, built-in ABP validation messages will not work.

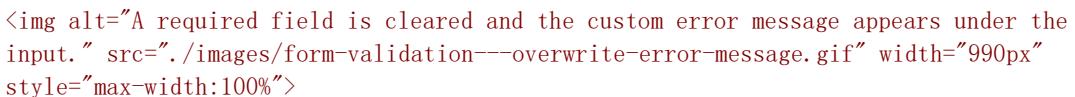
```

```js
import { VALIDATION_BLUEPRINTS } from "@ngx-validate/core";
import { DEFAULT_VALIDATION_BLUEPRINTS } from "@abp/ng.theme.shared";

@NgModule({
 providers: [
 {
 provide: VALIDATION_BLUEPRINTS,
 useValue: {
 ...DEFAULT_VALIDATION_BLUEPRINTS,
 required: "::RequiredInput",
 },
 },
 // other providers
],
 // rest of the module metadata
})
export class AppModule {}
```

```

The error message will look like this:



How to Disable Validation on a Form

If you want to validate a form manually, you can always disable automatic validation on it. All you need to do is place `skipValidation` on the form element.

```

```html
<form [FormGroup]="form" skipValidation>
 <!-- form fields here -->
</form>
```

```

How to Disable Validation on a Specific Field

Validation works on any element or component with a `formControl` or `formControlName` directive. You can disable automatic validation on a specific field by placing `skipValidation` on the input element or component.

```
```html
<input type="text" formControlName="name" skipValidation />
````
```

How to Use a Custom Error Component

First, build a custom error component. Extending the existing `ValidationErrorComponent` would make it easier.

```
```js
import { ValidationErrorComponent } from "@abp/ng.theme.basic";
import { ChangeDetectionStrategy, Component } from "@angular/core";

@Component({
 selector: "app-validation-error",
 template: `
 <div
 class="font-weight-bold font-italic px-1 invalid-feedback"
 *ngFor="let error of abpErrors; trackBy: trackByFn"
 >
 {{error.message | abpLocalization: error.interpolateParams}}
 </div>
 `,
 changeDetection: ChangeDetectionStrategy.OnPush,
})
export class ErrorComponent extends ValidationErrorComponent {}````
```

Then, declare and provide it in your root module.

```
```js
import { VALIDATION_ERROR_TEMPLATE } from "@ngx-validate/core";

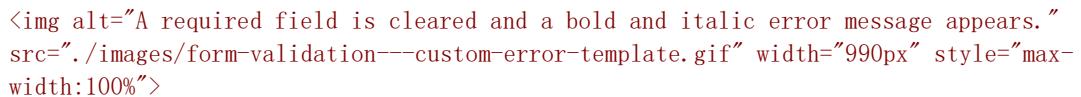
@NgModule({
  // rest of the module metadata

  declarations: [
    // other declarables
    ErrorComponent,
  ],
  providers: [
    // other providers
    {
      provide: VALIDATION_ERROR_TEMPLATE,
      useValue: ErrorComponent,
    },
  ],
})````
```

```
  ],
})
export class AppModule {
```

```

The error message will be bold and italic now:





#### 10.3.3.7 Settings

##### # Settings

You can get settings on the client-side using the [config state service](./Config-State.md) if they are allowed by their setting definition on the server-side.

➤ This document only explains how settings work in the Angular UI projects. See the [settings document](../../Settings.md) to understand the ABP setting system.

##### ## Before Use

To use the `ConfigStateService`, you must inject it in your class as a dependency. You do not have to provide the service explicitly, because it is already **\*\*provided in root\*\***.

```
```js
import { ConfigStateService } from '@abp/ng.core';

@Component({
  /* class metadata here */
})
class DemoComponent {
  constructor(private config: ConfigStateService) {}
}
```

```

##### ## How to Get a Specific Setting

You can use the `getSetting` method of `ConfigStateService` to get a specific setting from the configuration state. Here is an example:

```
```js
// this.config is instance of ConfigStateService

const defaultLang = this.config.getSetting("Abp.Localization.DefaultLanguage");
// 'en'
```

```

##### ### How to Get All Settings From the Store

You can use the `getSettings` method of `ConfigStateService` to obtain all settings as an object where the object properties are setting names and property values are setting values.

```
```js
// this.config is instance of ConfigStateService

const settings = this.config.getSettings();
// all settings as a key value pair
````
```

Additionally, the method lets you search settings by **\*\*passing a keyword\*\*** to it.

```
```js
const localizationSettings = this.config.getSettings("Localization");
/*
{
    'Abp.Localization.DefaultLanguage': 'en'
}
*/
````
```

Beware though, **\*\*settings search is case-sensitive\*\***.

#### 10.3.3.8 Features

##### # Features

You can get the value of a feature on the client-side using the [config state service](./Config-State.md) if it is allowed by the feature definition on the server-side.

> This document explains how to get feature values in an Angular application. See the [Features document](../../Features.md) to learn the feature system.

##### ## Before Use

To use the `ConfigStateService`, you must inject it in your class as a dependency. You do not have to provide the service explicitly, because it is already **\*\*provided in root\*\***.

```
```js
import { ConfigStateService } from '@abp/ng.core';

@Component({
    /* class metadata here */
})
class DemoComponent {
    constructor(private config: ConfigStateService) {}
}
````
```

## ## How to Get a Specific Feature

You can use the `getFeature` method of `ConfigStateService` to get a specific feature from the configuration state. Here is an example:

```
```js
// this.config is instance of ConfigStateService

const defaultLang = this.config.getFeature("Identity.TwoFactor");
// 'Optional'
````
```

You can then check the value of the feature to perform your logic. Please note that **\*\*feature keys are case-sensitive\*\***.

### 10.3.3.9 Global Features

#### # Angular: Global Features API

The `ConfigStateService.getGlobalFeatures` API allows you to get the enabled features of the [Global Features](../../Global-Features.md) on the client side.

➤ This document only explains the JavaScript API. See the [Global Features](../../Global-Features.md) document to understand the ABP Global Features system.

#### ## Usage

```
```js

import { ConfigStateService } from '@abp/ng.core';
import { Component, OnInit } from '@angular/core';

@Component({
    /* class metadata here */
})
class DemoComponent implements OnInit {
    constructor(private config: ConfigStateService) {}

    ngOnInit(): void {
        // Gets all enabled global features.
        const getGlobalFeatures = this.config.getGlobalFeatures();

        //Example result is: `{ enabledFeatures: [ 'Shopping.Payment',
        'Ecommerce.Subscription' ] }`

        // or
        this.config.getGlobalFeatures$().subscribe(getGlobalFeatures => {
            // use getGlobalFeatures here
        })
    }
}
```

```

// Check the global feature is enabled
this.config.getGlobalFeatureIsEnabled('Ecommerce.Subscription')

//Example result is `true`

this.config.getGlobalFeatureIsEnabled('My.Subscription')

//Example result is `false`

// or

this.config.getGlobalFeatureIsEnabled$('Ecommerce.Subscription').subscribe((isEnabled:boolean) => {
    // use isEnabled here
})
}
}
}

```

10.3.3.10 Permission Management

Permission Management

A permission is a simple policy that is granted or prohibited for a particular user, role or client. You can read more about [authorization in ABP](../../Authorization.md) document.

You can get permission of authenticated user using `getGrantedPolicy` or `getGrantedPolicy\$` method of `PermissionService`.

You can get permission as boolean value:

```

```js
import { PermissionService } from '@abp/ng.core';

export class YourComponent {
 constructor(private permissionService: PermissionService) {}

 ngOnInit(): void {
 const canCreate = this.permissionService.getGrantedPolicy('AbpIdentity.Roles.Create');
 }
}
```

```

You may also ****combine policy keys**** to fine tune your selection:

```

```js
// this.permissionService is instance of PermissionService

```

```

const hasIdentityAndAccountPermission = this.permissionService.getGrantedPolicy(
 "Abp.Identity && Abp.Account"
);

const hasIdentityOrAccountPermission = this.permissionService.getGrantedPolicy(
 "Abp.Identity || Abp.Account"
);
```

```

Please consider the following ****rules**** when creating your permission selectors:

- Maximum 2 keys can be combined.
- `&&` operator looks for both keys.
- `||` operator looks for either key.
- Empty string `''` as key will return `true`
- Using an operator without a second key will return `false`

Permission Directive

You can use the `PermissionDirective` to manage visibility of a DOM Element accordingly to user's permission.

```

```html
<div *abpPermission="`AbpIdentity.Roles`">
 This content is only visible if the user has 'AbpIdentity.Roles' permission.
</div>
```

```

As shown above you can remove elements from DOM with `abpPermission` structural directive.

Permission Guard

You can use `PermissionGuard` if you want to control authenticated user's permission to access to the route during navigation.

- * Import the PermissionGuard from @abp/ng.core.
- * Add `canActivate: [PermissionGuard]` to your route object.
- * Add `requiredPolicy` to the `data` property of your route in your routing module.

```

```js
import { PermissionGuard } from '@abp/ng.core';
// ...
const routes: Routes = [
 {
 path: 'path',
 component: YourComponent,
 canActivate: [PermissionGuard],
 data: {
 requiredPolicy: 'YourProjectName.YourComponent', // policy key for your component
 },
 },
]
```

```

```
];
```

Customization

In some cases, a custom permission management may be needed. All you need to do is to replace the service with your own. Here is how to achieve this:

- First, create a service of your own. Let's call it `CustomPermissionService` and extend `PermissionService` from `@abp/ng.core` as follows:

```
```js
import { ConfigStateService, PermissionService } from '@abp/ng.core';
import { Injectable } from '@angular/core';

@Injectable({
 providedIn: 'root',
})
export class CustomPermissionService extends PermissionService {
 constructor(configStateService: ConfigStateService) {
 super(configStateService);
 }

 // This is an example to show how to override the methods
 getGrantedPolicy$(key: string) {
 return super.getGrantedPolicy$(key);
 }
}
```

```

- Then, in `app.module.ts`, provide this service as follows:

```
```js
@NgModule({
 // ...
 providers: [
 // ...
 {
 provide: PermissionService,
 useExisting: CustomPermissionService,
 },
],
 // ...
})
export class AppModule {}
```

That's it. Now, when a directive/guard asks for `PermissionService` from angular, it will inject your service.

#### 10.3.3.11 Multi Tenancy

##### # Multi-Tenancy in Angular UI

ABP Angular UI supports the multi-tenancy. The following features related to multi-tenancy are available in the startup templates.

![Tenants Page](./images/tenants-page.png)

<p style="font-size:small;text-align:center;">Tenants page</p>

On the page above, you can;

- See all tenants.
- Create a new tenant.
- Edit an existing tenant.
- Delete a tenant.

![Tenant Switching Component](./images/tenant-switching-box.png)

<p style="font-size:small;text-align:center;">Tenant Switching Component</p>

You can switch between existing tenants by using the tenant switching box in the child pages of the MVC Account Public Module (like Login page). Angular UI gets selected tenant from `application-configuration` response and sends the tenant id to the backend as `\_\_tenant` header on each request.

##### ## Domain/Subdomain Tenant Resolver

> **\*\*Note:\*\*** If you are going to implement the steps below, you should also implement the domain/subdomain tenant resolver feature for the backend. See the [Domain/Subdomain Tenant Resolver section in Multi-Tenancy document](../../Multi-Tenancy#domain-subdomain-tenant-resolver) to learn the backend implementation.

Angular UI can get the tenant name from the app running URL. You can determine the current tenant by subdomain (like mytenant1.mydomain.com) or by the whole domain (like mytenant.com). To do this, you need to set the `application.baseUrl` property in the environment:

Subdomain resolver:

```
```js
// environment.prod.ts

export const environment = {
  ...
  application: {
    baseUrl: "https://{}.{mydomain}.com/",
  },
  ...
};
```

```
```
```

**\*\*{0}\*\*** is the placeholder to determine current tenant's unique name.

After the configuration above, if your app runs on the `mytenant1.mydomain.com`, the app will get the tenant name as **\*\*mytenant1\*\***. Then, the app will call the `/api/abp/multi-tenancy/tenants/by-name/mytenant1` endpoint to check if the tenant exists. If the tenant (mytenant1) exists, the app will keep this tenant data and send its `id` as `\_\_tenant` header to the backend on each request. If the tenant does not exist, the app will not send `\_\_tenant` header to the backend.

> **Important Note:** If you define the `baseUrl` with the placeholder (**\*\*{0}\*\***), the tenant switching component in the child pages of the `AccountLayoutComponent` (like Login page) will be hidden.

Domain resolver:

```
```js
// environment.prod.ts

export const environment = {
  ...
  application: {
    baseUrl: "https://{}.{}.com/",
  },
  ...
};

```
```

After the configuration above, if your app runs on the `mytenant.com`, the app will get the tenant name as **\*\*mytenant\*\***.

### ### Tenant Specific Remote Endpoints

The **\*\*{0}\*\*** placeholder can be put to the API URLs in the environment to determine tenant specific endpoints.

```
```js
// environment.prod.ts

export const environment = {
  ...
  application: {
    baseUrl: "https://{}.{}.mydomain.com/",
  },
  ...
},
oAuthConfig: {
  issuer: "https://{}.{}.ids.mydomain.com",
  ...
},
apis: {
  default: {
    ...
  }
};

```
```

```

 url: "https://{}.{0}.api.mydomain.com",
 },
 AbpIdentity: {
 url: "https://{}.{0}.identity.mydomain.com",
 },
},
};

```

```

> **Important Note:** The `application.baseUrl` and the `{0}` placeholder in the value of the `baseUrl` property are required to be able to get tenant from running URL. Other placeholders in API URLs are optional.

After the configuration above, if your app runs on the `mytenant1.mydomain.com`, the app will get tenant name as ****mytenant1**** and replace the environment object in `EnvironmentService` on app initialization as follows:

```

```js
// environment object in EnvironmentService

{
 //...
 application: {
 baseUrl: 'https://mytenant1.mydomain.com/',
 //...
 },
 oAuthConfig: {
 issuer: 'https://mytenant1.ids.mydomain.com',
 //...
 },
 apis: {
 default: {
 url: 'https://mytenant1.api.mydomain.com',
 },
 AbpIdentity: {
 url: 'https://mytenant1.identity.mydomain.com',
 },
 },
}
```

```

After this replacement, the app will use the following URLs:

- `https://mytenant1.ids.mydomain.com` as AuthServer URL.
- `https://mytenant1.api.mydomain.com` as default URL.
- `https://mytenant1.identity.mydomain.com` as `AbpIdentity` remote endpoint URL.

The app sends the `__tenant` header that contains the current tenant id on each request.

See Also

- [Multi Tenancy in ABP](../../../../Multi-Tenancy.md)

10.3.3.12 Account Module

Angular UI Account Module

Angular UI account module is available as of v4.3. It contains some pages (login, register, My account, etc.).

If you add the account module to your project;

- "My account" link in the current user dropdown on the top bar will redirect the user to a page in the account module.
- You can switch the authentication flow to the resource owner password flow.

Account Module Implementation

Install the `@abp/ng.account` NPM package by running the below command:

```
```bash
npm install @abp/ng.account
```
```

> Make sure v4.3 or higher version is installed.

Open the `app.module.ts` and add `AccountConfigModule.forRoot()` to the imports array as shown below:

```
```js
// app.module.ts

import { AccountConfigModule } from '@abp/ng.account/config';
//...

@NgModule({
 imports: [
 //...
 AccountConfigModule.forRoot()
],
 //...
})
export class AppModule {}
```
```

Open the `app-routing.module.ts` and add the `account` route to `routes` array as follows:

```
```js
// app-routing.module.ts
const routes: Routes = [
 //...
```
```

```

{
  path: 'account',
  loadChildren: () => import('@abp/ng.account').then(m => m.AccountModule.forRoot()),
},
//...
export class AppRoutingModule {}
```

```

### ### Account Public Module Implementation for Commercial Templates

The pro startup template comes with `@volo/abp.ng.account` package. You should update the package version to v4.3 or higher version. The package can be updated by running the following command:

```

```bash
npm install @volo/abp.ng.account
```

```

> Make sure v4.3 or higher version is installed.

Open the `app.module.ts` and add `AccountPublicConfigModule.forRoot()` to the imports array as shown below:

> Ensure that the `Account Layout Module` has been added if you are using the Lepton X theme. If you miss the step, you will get an error message that says `Account layout not found. Please check your configuration. If you are using LeptonX, please make sure you have added "AccountLayoutModule.forRoot()" to your app.module configuration.` when you try to access the account pages. Otherwise, you can skip adding the `AccountLayoutModule` step.

```

```js
// app.module.ts

import { AccountPublicConfigModule } from '@volo/abp.ng.account/public/config';
// if you are using or want to use Lepton X, you should add AccountLayoutModule
// import { AccountLayoutModule } from '@volosoft/abp.ng.theme.lepton-x/account'

//...

@NgModule({
  imports: [
    //...
    AccountPublicConfigModule.forRoot(),
    // AccountLayoutModule.forRoot() // Only for Lepton X
  ],
  //...
})
export class AppModule {}
```

```

Open the `app-routing.module.ts` and add the `account` route to `routes` array as follows:

```

```js
// app-routing.module.ts
const routes: Routes = [
  ...
  {
    path: 'account',
    loadChildren: () => import('@volo/abp.ng.account/public').then(m =>
      m.AccountPublicModule.forChild()),
  },
  ...
]
export class AppRoutingModule {}
```

```

#### ### My Account Page

Before v4.3, the "My account" link in the current user dropdown on the top bar redirected the user to MVC's profile management page. As of v4.3, if you added the account module to your project, the same link will land on a page in the Angular UI account module instead.

#### ### Personal Info Page Confirm Message

When the user changes their own data on the personal settings tab in My Account, The data can not update the CurrentUser key of Application-Configuration. The information of the user is stored in claims. The only way to apply this information to the CurrentUser of Application-Configuration is user should log out and log in. When the Refresh-Token feature is implemented, it will be fixed. So We've added a confirmation alert.

If you want to disable these warning, You should set  
`isPersonalSettingsChangedConfirmationActive` false

```

```js
// app-routing.module.ts
const routes: Routes = [
  ...
  {
    path: 'account',
    loadChildren: () => import('@volo/abp.ng.account/public').then(m =>
      m.AccountPublicModule.forChild({ isPersonalSettingsChangedConfirmationActive:false })),
  },
  ...
]
export class AppRoutingModule {}
```

```

#### ### Security Logs Page [COMMERCIAL]

Before v4.3, the "Security Logs" link in the current user dropdown on the top bar redirected the user to MVC's security logs page. As of v4.3, if you added the account module to your project, the same link will land on a page in the Angular UI account public module instead.

#### ### Resource Owner Password Flow

OAuth is preconfigured as authorization code flow in Angular application templates by default. If you added the account module to your project, you can switch the flow to resource owner password flow by changing the OAuth configuration in the `_environment.ts` files as shown below:

```
```js
import { Config } from '@abp/ng.core';

export const environment = {
    // other options removed for sake of brevity

    oAuthConfig: {
        issuer: 'https://localhost:44305', // AuthServer url
        clientId: 'MyProjectName_App',
        dummyClientSecret: '1q2w3e*',
        scope: 'offline_access MyProjectName',
    },
    // other options removed for sake of brevity
} as Config.Environment;
```

```

See the [\[Authorization in Angular UI\]](#) (`./Authorization.md`) document for more details.

#### 10.3.4 Utilities

##### 10.3.4.1 Managing RxJS Subscriptions

###### # Managing RxJS Subscriptions

``SubscriptionService`` is a utility service to provide an easy unsubscription from RxJS observables in Angular components and directives. Please see [\[why you should unsubscribe from observables on instance destruction\]](#) (<https://angular.io/guide/lifecycle-hooks#cleaning-up-on-instance-destruction>).

###### ## Getting Started

You have to provide the ``SubscriptionService`` at component or directive level, because it is **\*\*not provided in root\*\*** and it works in sync with component/directive lifecycle. Only after then you can inject and start using it.

```
```js
import { SubscriptionService } from '@abp/ng.core';

@Component({
    /* class metadata here */
    providers: [SubscriptionService],
})
class DemoComponent {
    count$ = interval(1000);
}
```

```

```

 constructor(private subscription: SubscriptionService) {
 this.subscription.addOne(this.count$, console.log);
 }
}
```

```

The values emitted by the `count\$` will be logged until the component is destroyed. You will not have to unsubscribe manually.

> Please do not try to use a singleton `SubscriptionService`. It simply will not work.

Usage

How to Subscribe to Observables

You can pass a `next` function and an `error` function.

```

```js
@Component({
 /* class metadata here */
 providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
 constructor(private subscription: SubscriptionService) {}

 ngOnInit() {
 const source$ = interval(1000);
 const nextFn = value => console.log(value * 2);
 const errorFn = error => {
 console.error(error);
 return of(null);
 };

 this.subscription.addOne(source$, nextFn, errorFn);
 }
}
```

```

Or, you can pass an observer.

```

```js
@Component({
 /* class metadata here */
 providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
 constructor(private subscription: SubscriptionService) {}

 ngOnInit() {
 const source$ = interval(1000);
 const observer = {
 next: value => console.log(value * 2),
 };
 this.subscription.addOne(source$, observer);
 }
}
```

```

```

        complete: () => console.log('DONE'),
    } ;

    this.subscription.addOne(source$, observer);
}
```

```

The ``addOne`` method returns the individual subscription, so that you may use it later on. Please see topics below for details.

#### ### How to Unsubscribe Before Instance Destruction

There are two ways to do that. If you are not going to subscribe again, you may use the ``closeAll`` method.

```

```js
@Component({
  /* class metadata here */
  providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
  constructor(private subscription: SubscriptionService) {}

  ngOnInit() {
    this.subscription.addOne(interval(1000), console.log);
  }

  onSomeEvent() {
    this.subscription.closeAll();
  }
}
```

```

This will clear all subscriptions, but you will not be able to subscribe again. If you are planning to add another subscription, you may use the ``reset`` method instead.

```

```js
@Component({
  /* class metadata here */
  providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
  constructor(private subscription: SubscriptionService) {}

  ngOnInit() {
    this.subscription.addOne(interval(1000), console.log);
  }

  onSomeEvent() {
    this.subscription.reset();
    this.subscription.addOne(interval(1000), console.warn);
  }
}
```

```

```
 }
}
```

```

How to Unsubscribe From a Single Subscription

Sometimes, you may need to unsubscribe from a particular subscription but leave others alive. In such a case, you may use the `closeOne` method.

```
```js
@Component({
 /* class metadata here */
 providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
 countSubscription: Subscription;

 constructor(private subscription: SubscriptionService) {}

 ngOnInit() {
 this.countSubscription = this.subscription.addOne(
 interval(1000),
 console.log
);
 }

 onSomeEvent() {
 this.subscription.closeOne(this.countSubscription);
 console.log(this.countSubscription.closed); // true
 }
}
```

```

How to Remove a Single Subscription From Tracked Subscriptions

You may want to take control of a particular subscription. In such a case, you may use the `removeOne` method to remove it from tracked subscriptions.

```
```js
@Component({
 /* class metadata here */
 providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
 countSubscription: Subscription;

 constructor(private subscription: SubscriptionService) {}

 ngOnInit() {
 this.countSubscription = this.subscription.addOne(
 interval(1000),
 console.log
);
 }

 onSomeEvent() {
 this.subscription.removeOne(this.countSubscription);
 console.log(this.countSubscription.closed); // false
 }
}
```

```

```

        );
    }

onSomeEvent() {
    this.subscription.removeOne(this.countSubscription);
    console.log(this.countSubscription.closed); // false
}
```

```

#### #### How to Check If Unsubscribed From All

Please use `isClosed` getter to check if `closeAll` was called before.

```

```js
@Component({
  /* class metadata here */
  providers: [SubscriptionService],
})
class DemoComponent implements OnInit {
  constructor(private subscription: SubscriptionService) {}

ngOnInit() {
  this.subscription.addOne(interval(1000), console.log);
}

onSomeEvent() {
  console.log(this.subscription.isClosed); // false
}
```

```

#### 10.3.4.2 Working with Lists

##### # Working with Lists

`ListService` is a utility service to provide easy pagination, sorting, and search implementation.

##### ## Getting Started

`ListService` is **\*\*not provided in root\*\***. The reason is, this way, it will clear any subscriptions on component destroy. You may use the optional `LIST\_QUERY\_DEBOUNCE\_TIME` token to adjust the debounce behavior.

```

```js
import { ListService } from '@abp/ng.core';
import { BookDto } from '../models';

```

```

import { BookService } from '../services';

@Component({
  /* class metadata here */
  providers: [
    // [Required]
    ListService,
    // [Optional]
    // Provide this token if you want a different debounce time.
    // Default is 300. Cannot be 0. Any value below 100 is not recommended.
    { provide: LIST_QUERY_DEBOUNCE_TIME, useValue: 500 },
  ],
  template: `
    `,
})
class BookComponent {
  items: BookDto[] = [];
  count = 0;

  constructor(
    public readonly list: ListService,
    private bookService: BookService,
  ) {
    // change ListService defaults here
    this.list.maxResultCount = 20;
  }

  ngOnInit() {
    // A function that gets query and returns an observable
    const bookStreamCreator = query => this.bookService.getList(query);

    this.list.hookToQuery(bookStreamCreator).subscribe(
      response => {
        this.items = response.items;
        this.count = response.count;
        // If you use OnPush change detection strategy,
        // call detectChanges method of ChangeDetectorRef here.
      }
    ); // Subscription is auto-cleared on destroy.
  }
}
```

```

> Noticed `list` is `public` and `readonly`? That is because we will use `ListService` directly in the component's template. That may be considered as an anti-pattern, but it is much quicker to implement. You can always use public component members to expose the `ListService` instance instead.

Bind `ListService` to ngx-datatable like this:

```
```html
<ngx-datatable
  [rows]="items"
  [count]="count"
  [list]="list"
  default
>
  <!-- column templates here -->
</ngx-datatable>
```

```

## Extending query with custom variables

You can extend the query parameter of the `ListService`'s `hookToQuery` method.

Firstly, you should pass your own type to `ListService` as shown below:

```
```typescript
constructor(public readonly list: ListService<BooksSearchParamsDto>) { }
```

```

Then update the `bookStreamCreator` constant like following:

```
```typescript
const bookStreamCreator = (query) => this.bookService.getList({...query, name: 'name
here'});
```

```

You can also create your params object.

Define a variable like this:

```
```typescript
booksSearchParams = {} as BooksSearchParamsDto;
```

```

Update the `bookStreamCreator` constant:

```
```typescript
const bookStreamCreator = (query) =>
this.bookService.getList({...query, ...this.booksSearchParams});
```

```

Then you can place inputs to the HTML:

```
```html
<div class="form-group">
  <input
    class="form control"
    placeholder="Name"
    (keyup.enter)="list.get()"
    [(ngModel)]="booksSearchParams.name"
  >
</div>
```

```

```
 />
</div>
````
```

`ListService` emits the `hookToQuery` stream when you call the `this.list.get()` method.

Usage with Observables

You may use observables in combination with [AsyncPipe](https://angular.io/guide/observables-in-angular#async-pipe) of Angular instead. Here are some possibilities:

```
```js
book$ = this.list.hookToQuery(query => this.bookService.getListByInput(query));
````
```

```
```html
<!-- simplified representation of the template -->
```

```
<ngx-datatable
[rows]="(book$ | async)?.items || []"
[count]=(book$ | async)?.totalCount || 0"
[list]="list"
default
>
 <!-- column templates here -->
</ngx-datatable>
```

```
<!-- DO NOT WORRY, ONLY ONE REQUEST WILL BE MADE -->
````
```

... or...

```
```js
@Select(BookState.getBooks)
books$: Observable<BookDto[]>;

@Select(BookState.getBookCount)
bookCount$: Observable<number>;

ngOnInit() {
 this.list.hookToQuery((query) => this.store.dispatch(new
GetBooks(query))).subscribe();
}
````
```

```
```html
<!-- simplified representation of the template -->
```

```
<ngx-datatable
[rows]=(books$ | async) || []"
```

```
[count]="${bookCount$ | async} || 0"
[list]="list"
default
>
<!-- column templates here -->
</ngx-datatable>
```

```

› We do not recommend using the NGXS store for CRUD pages unless your application needs to share list information between components or use it later on in another page.

How to Refresh Table on Create/Update/Delete

`ListService` exposes a `get` method to trigger a request with the current query. So, basically, whenever a create, update, or delete action resolves, you can call `this.list.get();` and it will call hooked stream creator again.

```
```js
this.bookService.createByInput(form.value)
 .subscribe(() => {
 this.list.get();

 // Other subscription logic here
 });
```

```

...or...

```
```js
this.store.dispatch(new DeleteBook(id)).subscribe(this.list.get);
```

```

› We do not recommend using the NGXS store for CRUD pages unless your application needs to share list information between components or use it later on in another page.

How to Implement Server-Side Search in a Table

`ListService` exposes a `filter` property that will trigger a request with the current query and the given search string. All you need to do is to bind it to an input element with two-way binding.

```
```html
<!-- simplified representation -->

<input type="text" name="search" [(ngModel)]="list.filter">
```

```

ABP doesn't have a built-in filtering mechanism. You need to implement it yourself and handle the `filter` property in the backend.

10.3.4.3 Easy *ngFor trackBy

```
# Easy *ngFor trackBy
```

`TrackByService` is a utility service to provide an easy implementation for one of the most frequent needs in Angular templates: `TrackByFunction`. Please see [[this page in Angular docs](#)](<https://angular.io/guide/template-syntax#ngfor-with-trackby>) for its purpose.

Getting Started

You do not have to provide the `TrackByService` at module or component level, because it is already **provided in root****. You can inject and start using it immediately in your components. For better type support, you may pass in the type of the iterated item to it.

```
```js
import { TrackByService } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 list: Item[];

 constructor(public readonly track: TrackByService<Item>) {}

}
```

```

> Noticed `track` is `public` and `readonly`? That is because we will see some examples where methods of `TrackByService` instance are directly called in the component's template. That may be considered as an anti-pattern, but it has its own advantage, especially when component inheritance is leveraged. You can always use public component properties instead.

****The members are also exported as separate functions.**** If you do not want to inject `TrackByService`, you can always import and use those functions directly in your classes.

Usage

There are two approaches available.

1. You may inject `TrackByService` to your component and use its members.
2. You may use exported higher-order functions directly on component properties.

How to Track Items by a Key

You can use `by` to get a `TrackByFunction` that tracks the iterated object based on one of its keys. For type support, you may pass in the type of the iterated item to it.

```
```html
<!-- template of DemoComponent -->

<div *ngFor="let item of list; trackBy: track.by('id')">{{item.name}}</div>
```

```

`by` is exported as a stand-alone function and is named `trackBy`.

```
```js
import { trackBy } from "@abp/ng.core";

@Component({
 template: `
 <div
 *ngFor="let item of list; trackBy: trackById"
 >
 {{item.name}}
 </div>
 `,
})
class DemoComponent {
 list: Item[];
 trackById = trackBy<Item>('id');
}
```

```

How to Track by a Deeply Nested Key

You can use `byDeep` to get a `TrackByFunction` that tracks the iterated object based on a deeply nested key. For type support, you may pass in the type of the iterated item to it.

```
```html
<!-- template of DemoComponent -->

<div
 *ngFor="let item of list; trackBy: track.byDeep('tenant', 'account', 'id')"
>
 {{item.tenant.name}}
</div>
```

```

```

`byDeep` is exported as a stand-alone function and is named `trackByDeep`.

```
```js
import { trackByDeep } from "@abp/ng.core";

@Component({
  template: `
    <div
      *ngFor="let item of list; trackBy: trackByTenantAccountId"
    >
      {{item.name}}
    </div>
  `,
})
class DemoComponent {
  list: Item[];
}

trackByTenantAccountId = trackByDeep<Item>('tenant', 'account', 'id');
```
```

```

10.3.4.4 Router Events

```
# Router Events Simplified
```

`RouterEvents` is a utility service for filtering specific router events and reacting to them. Please see [this page in Angular docs](<https://angular.io/api/router/Event>) for available router events.

```
## Benefit
```

You can use router events directly and filter them as seen below:

```
```js
import {
 NavigationEnd,
 NavigationError,
 NavigationCancel,
 Router,
} from '@angular/router';
import { filter } from 'rxjs/operators';

@Injectable()

```

```

class SomeService {
 navigationFinish$ = this.router.events.pipe(
 filter(
 event =>
 event instanceof NavigationEnd ||
 event instanceof NavigationError ||
 event instanceof NavigationCancel,
),
);
 /* Observable<Event> */

 constructor(private router: Router) {}
}
```

```

However, `RouterEvents` makes filtering router events easier.

```

```js
import { RouterEvents } from '@abp/ng.core';

@Injectable()
class SomeService {
 navigationFinish$ = this.routerEvents.getNavigationEvents('End', 'Error', 'Cancel');
 /* Observable<NavigationCancel | NavigationEnd | NavigationError> */

 constructor(private routerEvents: RouterEvents) {}
}
```

```

`RouterEvents` also delivers improved type-safety. In the example above, `navigationFinish\$` has inferred type of `Observable<NavigationCancel | NavigationEnd | NavigationError>` whereas it would have `Observable<Event>` when router events are filtered directly.

Usage

You do not have to provide `RouterEvents` at the module or component level, because it is already **provided in root**. You can inject and start using it immediately in your components.

How to Get Specific Navigation Events

You can use `getNavigationEvents` to get a stream of navigation events matching given event keys.

```

```js
import { RouterEvents } from '@abp/ng.core';
import { merge } from 'rxjs';

```

```

import { mapTo } from 'rxjs/operators';

@Injectable()
class SomeService {
 navigationStart$ = this.routerEvents.getNavigationEvents('Start');
 /* Observable<NavigationStart> */

 navigationFinish$ = this.routerEvents.getNavigationEvents('End', 'Error', 'Cancel');
 /* Observable<NavigationCancel | NavigationEnd | NavigationError> */

 loading$ = merge(
 this.navigationStart$.pipe(mapTo(true)),
 this.navigationFinish$.pipe(mapTo(false)),
);
 /* Observable<boolean> */

 constructor(private routerEvents: RouterEvents) {}
}
```

```

How to Get All Navigation Events

You can use `getAllNavigationEvents` to get a stream of all navigation events without passing any keys.

```

```js
import { RouterEvents, NavigationStart } from '@abp/ng.core';
import { map } from 'rxjs/operators';

@Injectable()
class SomeService {
 navigationEvent$ = this.routerEvents.getAllNavigationEvents();
 /* Observable<NavigationCancel | NavigationEnd | NavigationError | NavigationStart> */

 loading$ = this.navigationEvent$.pipe(
 map(event => event instanceof NavigationStart),
);
 /* Observable<boolean> */

 constructor(private routerEvents: RouterEvents) {}
}
```

```

How to Get Specific Router Events

You can use `getEvents` to get a stream of router events matching given event constructors.

```

```js
import { RouterEvents } from '@abp/ng.core';

```

```

import { ActivationEnd, ChildActivationEnd } from '@angular/router';

@Injectable()
class SomeService {
 moduleActivation$ = this.routerEvents.getEvents(ActivationEnd, ChildActivationEnd);
 /* Observable<ActivationEnd | ChildActivationEnd> */

 constructor(private routerEvents: RouterEvents) {}
}
```

```

How to Get All Router Events

You can use `getEvents` to get a stream of all router events without passing any event constructors. This is nothing different from accessing `events` property of `Router` and is added to the service just for convenience.

```

```js
import { RouterEvents } from '@abp/ng.core';
import { ActivationEnd, ChildActivationEnd } from '@angular/router';

@Injectable()
class SomeService {
 routerEvent$ = this.routerEvents.getAllEvents();
 /* Observable<Event> */

 constructor(private routerEvents: RouterEvents) {}
}
```

```

10.3.4.5 Inserting Scripts & Styles to DOM

Inserting Scripts & Styles to DOM

You can use the `DomInsertionService` in @abp/ng.core package in order to insert scripts and styles in an easy and explicit way.

Getting Started

You do not have to provide the `DomInsertionService` at module or component level, because it is already **provided in root****. You can inject and start using it immediately in your components, directives, or services.

```

```js
import { DomInsertionService } from '@abp/ng.core';

@Component({
 /* class metadata here */
}

```

```
)
class DemoComponent {
 constructor(private domInsertionService: DomInsertionService) {}
}
```
```

Usage

You can use the `insertContent` method of `DomInsertionService` to create a `

How to Insert Scripts

The first parameter of `'insertContent'` method expects a `'ContentStrategy'`. If you pass a `'ScriptContentStrategy'` instance, the `'DomInsertionService'` will create a `'<script>'` element with given `'content'` and place it in the designated DOM position.

```
```js
import { DomInsertionService, CONTENT_STRATEGY } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 constructor(private domInsertionService: DomInsertionService) {}

 ngOnInit() {
 const scriptElement = this.domInsertionService.insertContent(
 CONTENT_STRATEGY.AppendScriptToBody('alert()')
);
 }
}
```

In the example above, `<script>alert()</script>` element will place at the **\*\*end\*\*** of `<body>` and `scriptElement` will be an `HTMLScriptElement`.

Please refer to [\[ContentStrategy\]](#) (`./Content-Strategy.md`) to see all available content strategies and how you can build your own content strategy.

> Important Note: `DomInsertionService` does not insert the same content twice. In order to add a content again, you first should remove the old content using `removeContent` method.

## ### How to Insert Styles

If you pass a `StyleContentStrategy` instance as the first parameter of `insertContent` method, the `DomInsertionService` will create a `<style>` element with given `content` and place it in the designated DOM position.

js

```

import { DomInsertionService, CONTENT_STRATEGY } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 constructor(private domInsertionService: DomInsertionService) {}

ngOnInit() {
 const styleElement = this.domInsertionService.insertContent(
 CONTENT_STRATEGY.AppendStyleToHead('body {margin: 0;}')
);
}
...
```

```

In the example above, `<style>body {margin: 0;}</style>` element will place at the ****end**** of `<head>` and `styleElement` will be an `HTMLStyleElement`.

Please refer to [ContentStrategy](./Content-Strategy.md) to see all available content strategies and how you can build your own content strategy.

> Important Note: `DomInsertionService` does not insert the same content twice. In order to add a content again, you first should remove the old content using `removeContent` method.

How to Remove Inserted Scripts & Styles

If you pass the inserted `HTMLScriptElement` or `HTMLStyleElement` element as the first parameter of `removeContent` method, the `DomInsertionService` will remove the given element.

```

```js
import { DomInsertionService, CONTENT_STRATEGY } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 private styleElement: HTMLStyleElement;

 constructor(private domInsertionService: DomInsertionService) {}

ngOnInit() {
 this.styleElement = this.domInsertionService.insertContent(
 CONTENT_STRATEGY.AppendStyleToHead('body {margin: 0;}')
);
}

ngOnDestroy() {
 this.domInsertionService.removeContent(this.styleElement);
}
```

```

```
}
```

In the example above, `<style>body {margin: 0;}</style>` element **will be removed** from `<head>` when the component is destroyed.

```
## API
```

```
### insertContent
```

```
```js
insertContent<T extends HTMLScriptElement | HTMLStyleElement>(
 contentStrategy: ContentStrategy<T>,
): T
```

```

- `contentStrategy` parameter is the primary focus here and is explained above.
- returns `HTMLScriptElement` or `HTMLStyleElement` based on given strategy.

```
### removeContent
```

```
```js
removeContent(element: HTMLScriptElement | HTMLStyleElement): void
```

```

- `element` parameter is the inserted `HTMLScriptElement` or `HTMLStyleElement` element, which was returned by `insertContent` method.

```
### has
```

```
```js
has(content: string): boolean
```

```

The `has` method returns a boolean value that indicates the given content has already been added to the DOM or not.

- `content` parameter is the content of the inserted `HTMLScriptElement` or `HTMLStyleElement` element.

10.3.4.6 Lazy Loading Scripts & Styles

```
# Lazy Loading Scripts & Styles
```

You can use the `LazyLoadService` in @abp/ng.core package in order to lazy load scripts and styles in an easy and explicit way.

Getting Started

You do not have to provide the `LazyLoadService` at module or component level, because it is already **provided in root**. You can inject and start using it immediately in your components, directives, or services.

```
```js
import { LazyLoadService } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 constructor(private lazyLoadService: LazyLoadService) {}
}
```

```

Usage

You can use the `load` method of `LazyLoadService` to create a `

The `load` method returns an observable to which you can subscribe in your component or with an `async` pipe. In the example above, the `NgIf` directive will render `<some-component>` only ****if the script gets successfully loaded or is already loaded before****.

- You can subscribe multiple times in your template with `async` pipe. The Scripts will only be loaded once.

Please refer to [LoadingStrategy](./Loading-Strategy.md) to see all available loading strategies and how you can build your own loading strategy.

How to Load Styles

If you pass a `StyleLoadingStrategy` instance as the first parameter of `load` method, the `LazyLoadService` will create a `<link>` element with given `href` and place it in the designated DOM position.

```
```js
import { LazyLoadService, LOADING_STRATEGY } from '@abp/ng.core';

@Component({
 template: `
 <some-component *ngIf="stylesLoaded$ | async"></some-component>
 `
})
class DemoComponent {
 stylesLoaded$ = this.lazyLoad.load(
 LOADING_STRATEGY.AppendAnonymousStyleToHead('/assets/some-styles.css'),
);

 constructor(private lazyLoadService: LazyLoadService) {}
}
````
```

The `load` method returns an observable to which you can subscribe in your component or with an `AsyncPipe`. In the example above, the `NgIf` directive will render `<some-component>` only ****if the style gets successfully loaded or is already loaded before****.

- You can subscribe multiple times in your template with `async` pipe. The styles will only be loaded once.

Please refer to [LoadingStrategy](./Loading-Strategy.md) to see all available loading strategies and how you can build your own loading strategy.

Advanced Usage

You have quite a bit of ****freedom to define how your lazy load will work****. Here is an example:

```

```js
const domStrategy = DOM_STRATEGY.PrependToHead();

const crossOriginStrategy = CROSS_ORIGIN_STRATEGY.Anonymous(
 'sha384-Vkoo8x4CGsO3+Hhxv8T/Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh',
);

const loadingStrategy = new StyleLoadingStrategy(
 'https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css',
 domStrategy,
 crossOriginStrategy,
);

this.lazyLoad.load(loadingStrategy, 1, 2000);
```

```

This code will create a `<link>` element with given url and integrity hash, insert it to top of the `<head>` element, and retry once after 2 seconds if first try fails.

A common usecase is ****loading multiple scripts and/or styles before using a feature****:

```

```js
import { LazyLoadService, LOADING_STRATEGY } from '@abp/ng.core';
import { forkJoin } from 'rxjs';

@Component({
 template: `
 <some-component *ngIf="scriptsAndStylesLoaded$ | async"></some-component>
 `
})
class DemoComponent {
 private stylesLoaded$ = forkJoin(
 this.lazyLoad.load(
 LOADING_STRATEGY.PrependAnonymousStyleToHead('/assets/library-dark-theme.css'),
),
 this.lazyLoad.load(
 LOADING_STRATEGY.PrependAnonymousStyleToHead('/assets/library.css'),
),
);

 private scriptsLoaded$ = forkJoin(
 this.lazyLoad.load(
 LOADING_STRATEGY.AppendAnonymousScriptToHead('/assets/library.js'),
),
 this.lazyLoad.load(
 LOADING_STRATEGY.AppendAnonymousScriptToHead('/assets/other-library.css'),
),
);

 scriptsAndStylesLoaded$ = forkJoin(this.scriptsLoaded$, this.stylesLoaded$);
}

```

```
 constructor(private lazyLoadService: LazyLoadService) {}
}
```

```

RxJS `forkJoin` will load all scripts and styles in parallel and emit only when all of them are loaded. So, when ``<some-component>`` is placed, all required dependencies will be available.

➤ Noticed we have prepended styles to the document head? This is sometimes necessary, because your application styles may be overriding some of the library styles. In such a case, you must be careful about the order of prepended styles. They will be placed one-by-one and, ****when prepending, the last one placed will be on top****.

Another frequent usecase is ****loading dependent scripts in order****:

```
```js
import { LazyLoadService, LOADING_STRATEGY } from '@abp/ng.core';
import { concat } from 'rxjs';

@Component({
 template: `
 <some-component *ngIf="scriptsLoaded$ | async"></some-component>
 `
})
class DemoComponent {
 scriptsLoaded$ = concat(
 this.lazyLoad.load(
 LOADING_STRATEGY.PrependAnonymousScriptToHead('/assets/library.js'),
),
 this.lazyLoad.load(
 LOADING_STRATEGY.AppendAnonymousScriptToHead('/assets/script-that-requires-
library.js'),
),
);
}

constructor(private lazyLoadService: LazyLoadService) {}
```

```

In this example, the second file needs the first one to be loaded beforehand. RxJS `concat` function will let you load all scripts one-by-one in the given order and emit only when all of them are loaded.

API

loaded

```
```js
loaded: Set<string>
````
```

All previously loaded paths are available via this property. It is a simple [JavaScript Set](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Set).

load

```
```js
load(strategy: LoadingStrategy, retryTimes?: number, retryDelay?: number): Observable<Event>
````
```

- `strategy` parameter is the primary focus here and is explained above.
- `retryTimes` defines how many times the loading will be tried again before fail (`_default: 2`).
- `retryDelay` defines how much delay there will be between retries (`_default: 1000`).

10.3.4.7 Projecting Angular Content

Projecting Angular Content

You can use the `ContentProjectionService` in @abp/ng.core package in order to project content in an easy and explicit way.

Getting Started

You do not have to provide the `ContentProjectionService` at module or component level, because it is already **provided in root**. You can inject and start using it immediately in your components, directives, or services.

```
```js
import { ContentProjectionService } from '@abp/ng.core';

@Component({
 /* class metadata here */
})
class DemoComponent {
 constructor(private contentProjectionService: ContentProjectionService) {}
}
````
```

Usage

You can use the `projectContent` method of `ContentProjectionService` to render components and templates dynamically in your project.

How to Project Components to Root Level

If you pass a `RootComponentProjectionStrategy` as the first parameter of `projectContent` method, the `ContentProjectionService` will resolve the projected component and place it at the root level. If provided, it will also pass the component a context.

```
```js
const strategy = PROJECTION_STRATEGY.AppendComponentToBody(
 SomeOverlayComponent,
 { someOverlayProp: "SOME_VALUE" }
);

const componentRef = this.contentProjectionService.projectContent(strategy);
````
```

In the example above, `SomeOverlayComponent` component will placed at the ****end**** of `<body>` and a `ComponentRef` will be returned. Additionally, the given context will be applied, so `someOverlayProp` of the component will be set to `SOME_VALUE`.

- You should keep the returned `ComponentRef` instance, as it is a reference to the projected component and you will need that reference to destroy the projected view and the component instance.

How to Project Components and Templates into a Container

If you pass a `ComponentProjectionStrategy` or `TemplateProjectionStrategy` as the first parameter of `projectContent` method, and a `ViewContainerRef` as the second parameter of that strategy, the `ContentProjectionService` will project the component or template to the given container. If provided, it will also pass the component or the template a context.

```
```js
const strategy = PROJECTION_STRATEGY.ProjectComponentToContainer(
 SomeComponent,
 viewContainerRefOfTarget,
 { someProp: "SOME_VALUE" }
);

const componentRef = this.contentProjectionService.projectContent(strategy);
````
```

In this example, the `viewContainerRefOfTarget`, which is a `ViewContainerRef` instance, will be cleared and `SomeComponent` component will be placed inside it. In addition, the given context will be applied and `someProp` of the component will be set to `SOME_VALUE`.

- You should keep the returned `ComponentRef` or `EmbeddedViewRef`, as they are a reference to the projected content and you will need them to destroy it when necessary.

Please refer to [ProjectionStrategy] (./Projection-Strategy.md) to see all available projection strategies and how you can build your own projection strategy.

API

```
#### projectContent

```js
projectContent<T extends Type<any> | TemplateRef<any>>(
 projectionStrategy: ProjectionStrategy<T>,
 injector = this.injector,
): ComponentRef<C> | EmbeddedViewRef<C>
```

```

- `projectionStrategy` parameter is the primary focus here and is explained above.
- `injector` parameter is the `Injector` instance you can pass to the projected content. It is not used in `TemplateProjectionStrategy`.

10.3.4.8 Modal

Modal

`ModalComponent` is a pre-built component exposed by `@abp/ng.theme.shared` package to show modals. The component uses the [`ng-bootstrap`] (<https://ng-bootstrap.github.io/>)'s modal service inside to render a modal.

The `abp-modal` provides some additional benefits:

- It is **flexible**. You can pass header, body, footer templates easily by adding the templates to the `abp-modal` content. It can also be implemented quickly.
- Provides several inputs be able to customize the modal and several outputs be able to listen to some events.
- Automatically detects the close button which has a `abpClose` directive attached to and closes the modal when pressed this button.
- Automatically detects the `abp-button` and triggers its loading spinner when the `busy` input value of the modal component is true.
- Automatically checks if the form inside the modal **has changed, but not saved**. It warns the user by displaying a [confirmation popup] (Confirmation-Service) in this case when a user tries to close the modal or refresh/close the tab of the browser.

> Note: A modal can also be rendered by using the `ng-bootstrap` modal. For further information, see [Modal doc] (<https://ng-bootstrap.github.io/#/components/modal>) on the `ng-bootstrap` documentation.

Getting Started

In order to use the `abp-modal` in an HTML template, the **`ThemeSharedModule`** should be imported into your module like this:

```

```js
// ...
import { ThemeSharedModule } from '@abp/ng.theme.shared';

@NgModule({
 //...
 imports: [..., ThemeSharedModule],
})
export class MyFeatureModule {}
```

```

Usage

You can add the `abp-modal` to your component very quickly. See an example:

```

```html
<!-- sample.component.html -->

<button class="btn btn-primary" (click)="isModalOpen = true">Open modal</button>

<abp-modal [(visible)]="isModalOpen">
 <ng-template #abpHeader>
 <h3>Modal Title</h3>
 </ng-template>

 <ng-template #abpBody>
 <p>Modal content</p>
 </ng-template>

 <ng-template #abpFooter>
 <button type="button" class="btn btn-secondary" abpClose>Close</button>
 </ng-template>
</abp-modal>
```

```

```

```js
// sample.component.ts

@Component(/* component metadata */)
export class SampleComponent {
 isModalOpen = false
}

![Example modal result] ./images/modal-result-1.jpg

```

See an example form inside a modal:

```

```html
<!-- book.component.ts -->

```

```

<abp-modal [(visible)]="isModalOpen" [busy]="inProgress">
  <ng-template #abpHeader>
    <h3>Book</h3>
  </ng-template>

  <ng-template #abpBody>
    <form id="book-form" [FormGroup]="form" (ngSubmit)="save()">
      <div class="form-group">
        <label for="book-name">Author</label><span> * </span>
        <input type="text" id="author" class="form-control" formControlName="author" autofocus />
      </div>

      <div class="form-group">
        <label for="book-name">Name</label><span> * </span>
        <input type="text" id="book-name" class="form-control" formControlName="name" />
      </div>

      <div class="form-group">
        <label for="book-price">Price</label><span> * </span>
        <input type="number" id="book-price" class="form-control" formControlName="price" />
      </div>

      <div class="form-group">
        <label for="book-type">Type</label><span> * </span>
        <select class="form-control" id="book-type" formControlName="type">
          <option [ngValue]="">Select a book type</option>
          <option [ngValue]="">Undefined</option>
          <option [ngValue]="">Adventure</option>
          <option [ngValue]="">Biography</option>
          <option [ngValue]="">Fantastic</option>
          <option [ngValue]="">Science</option>
        </select>
      </div>

      <div class="form-group">
        <label for="book-publish-date">Publish date</label><span> * </span>
        <input
          id="book-publish-date"
          formControlName="publishDate"
          class="form-control"
          type="date"
        />
      </div>
    </form>
  </ng-template>

  <ng-template #abpFooter>
    <button type="button" class="btn btn-secondary" abpClose>
      Cancel
    </button>
  
```

```

<button form="book-form" class="btn btn-primary" [disabled]="form.invalid || form.pristine">
    <i class="fa fa-check mr-1"></i>
    Save
</button>
</ng-template>
</abp-modal>
```
```
ts
// book.component.ts

import { Component } from '@angular/core';
import { FormBuilder, Validators } from '@angular/forms';

@Component({/* component metadata */})
export class BookComponent {
  form = this.fb.group({
    author: [null, [Validators.required]],
    name: [null, [Validators.required]],
    price: [null, [Validators.required, Validators.min(0)]],
    type: [null, [Validators.required]],
    publishDate: [null, [Validators.required]],
  });

  inProgress: boolean;

  isModalOpen: boolean;

  constructor(private fb: FormBuilder, private service: BookService) {}

  save() {
    if (this.form.invalid) return;

    this.inProgress = true;

    this.service.save(this.form.value).subscribe(() => {
      this.inProgress = false;
    });
  }
}
```

```

The modal with form looks like this:

![Form example result](./images/modal-result-2.jpg)

## API

### Inputs

```
visible
```

```
```js
@Input() visible: boolean
```
```

**\*\*`visible`\*\*** is a boolean input that determines whether the modal is open. It is also can be used two-way binding.

```
busy
```

```
```js
@Input() busy: boolean
```
```

**\*\*`busy`\*\*** is a boolean input that determines whether the busy status of the modal is true. When `busy` is true, the modal cannot be closed and the `abp-button` loading spinner is triggered.

```
options
```

```
```js
@Input() options: NgbModalOptions
```
```

**\*\*`options`\*\*** is an input typed [NgbModalOptions](https://ng-bootstrap.github.io/#/components/modal/api#NgbModalOptions). It is configuration for the `ng-bootstrap` modal.

```
suppressUnsavedChangesWarning
```

```
```js
@Input() suppressUnsavedChangesWarning: boolean
```
```

**\*\*`suppressUnsavedChangesWarning`\*\*** is a boolean input that determines whether the confirmation popup triggering active or not. It can also be set globally as shown below:

```
```ts
//app.module.ts

// app.module.ts

import { SUPPRESS_UNSAVED_CHANGES_WARNING } from '@abp/ng.theme.shared';

// ...

@NgModule({
  // ...
  providers: [{provide: SUPPRESS_UNSAVED_CHANGES_WARNING, useValue: true}]
})
```

```
export class AppModule {}  
```
```

Note: The `suppressUnsavedChangesWarning` input of `abp-modal` value overrides the `SUPPRESS\_UNSAVED\_CHANGES\_WARNING` injection token value.

### Outputs

#### visibleChange

```
```js  
@Output() readonly visibleChange = new EventEmitter<boolean>();  
```
```

\*\*`visibleChange`\*\* is an event emitted when the modal visibility has changed. The event payload is a boolean.

#### appear

```
```js  
@Output() readonly appear = new EventEmitter<void>();  
```
```

\*\*`appear`\*\* is an event emitted when the modal has opened.

#### disappear

```
```js  
@Output() readonly disappear = new EventEmitter<void>();  
```
```

\*\*`disappear`\*\* is an event emitted when the modal has closed.

#### 10.3.4.9 Confirmation Popup

```
Confirmation Popup
```

You can use the `ConfirmationService` in @abp/ng. theme.shared package to display a confirmation popup by placing at the root level in your project.

## Getting Started

You do not have to provide the `ConfirmationService` at module or component level, because it is already \*\*provided in root\*\*. You can inject and start using it immediately in your components, directives, or services.

```
```js  
import { ConfirmationService } from '@abp/ng. theme.shared';  
  
@Component({
```

```
/* class metadata here */
})
class DemoComponent {
  constructor(private confirmation: ConfirmationService) {}
}
```

```

#### ## Usage

You can use the `success`, `warn`, `error`, and `info` methods of `ConfirmationService` to display a confirmation popup.

#### #### How to Display a Confirmation Popup

```
```js
const confirmationStatus$ = this.confirmation.success("Message", "Title");
```

```

- The `ConfirmationService` methods accept three parameters that are `message`, `title`, and `options`.
- `success`, `warn`, `error`, and `info` methods return an [RxJS Subject](<https://rxjs-dev.firebaseio.com/guide/subject>) to listen to confirmation popup closing event. The type of event value is [`Confirmation.Status`](<https://github.com/abpframework/abp/blob/master/npm/ng-packs/packages/theme-shared/src/lib/models/confirmation.ts#L24>) that is an enum.

#### #### How to Listen Closing Event

You can subscribe to the confirmation closing event like below:

```
```js
import { Confirmation, ConfirmationService } from '@abp/ng.theme.shared';

constructor(private confirmation: ConfirmationService) {}

this.confirmation
  .warn('::WillBeDeleted', { key: '::AreYouSure', defaultValue: 'Are you sure?' })
  .subscribe((status: Confirmation.Status) => {
    // your code here
  });
```

```

- The `message` and `title` parameters accept a string, localization key or localization object. See the [localization document](./Localization.md)
- `Confirmation.Status` is an enum and has three properties;
  - `Confirmation.Status.confirm` is a closing event value that will be emitted when the popup is closed by the confirm button.
  - `Confirmation.Status.reject` is a closing event value that will be emitted when the popup is closed by the cancel button.
  - `Confirmation.Status.dismiss` is a closing event value that will be emitted when the popup is closed by pressing the escape or clicking the backdrop.

If you are not interested in the confirmation status, you do not have to subscribe to the returned observable:

```
```js
this.confirmation.error("You are not authorized.", "Error");
```
```

#### #### How to Display a Confirmation Popup With Given Options

Options can be passed as the third parameter to `success`, `warn`, `error`, and `info` methods:

```
```js
const options: Partial<Confirmation.Options> = {
  hideCancelBtn: false,
  hideYesBtn: false,
  dismissible: false,
  cancelText: "Close",
  yesText: "Confirm",
  messageLocalizationParams: ["Demo"],
  titleLocalizationParams: [],
  // You can customize icon
  // icon: 'fa fa-exclamation-triangle', // or
  // iconTemplate : ''
}

this.confirmation.warn(
  "AbpIdentity::RoleDeletionConfirmationMessage",
  "Are you sure?",
  options
);
```
```

- `hideCancelBtn` option hides the cancellation button when `true`. Default value is `false`.
- `hideYesBtn` option hides the confirmation button when `true`. Default value is `false`.
- `dismissible` option allows dismissing the confirmation popup by pressing escape or clicking the backdrop. Default value is `true`.
- `cancelText` is the text of the cancellation button. A localization key or localization object can be passed. Default value is `AbpUi::Cancel`.
- `yesText` is the text of the confirmation button. A localization key or localization object can be passed. Default value is `AbpUi::Yes`.
- `messageLocalizationParams` is the interpolation parameters for the localization of the message.
- `titleLocalizationParams` is the interpolation parameters for the localization of the title.
- `icon` is the custom class of the icon. Default value is `undefined`.
- `iconTemplate` is the template for icon. Default value is `undefined`.

With the options above, the confirmation popup looks like this:

```
![confirmation](./images/confirmation.png)
```

You are able to pass in an HTML string as title, message, or button texts. Here is an example:

```
```js
const options: Partial<Confirmation.Options> = {
  yesText: '<i class="fa fa-trash mr-1"></i>Yes, delete it',
};

this.confirmation.warn(
  `
    <strong>Role Demo</strong> will be <strong>deleted</strong>
    <br>
    Do you confirm that?
  `,
  '<span class="my-custom-title">Are you sure?</span>',
  options
);
```

```

Since the values are HTML now, localization should be handled manually. Check out the [\[LocalizationService\]](#)(./Localization#using-the-localization-service) to see how you can accomplish that.

➤ Please note that all strings will be sanitized by Angular and not every HTML string will work. Only values that are considered as "safe" by Angular will be displayed.

#### #### How to Remove a Confirmation Popup

The open confirmation popup can be removed manually via the `clear` method:

```
```js
this.confirmation.clear();
```

```

#### #### How to Change Icons of The Confirmation Popup

You can change icons with the token of "confirmationIcons" in ThemeSharedModule in the app.module.ts. The changes will affect all confirmation popup in the project.

```
```js
...
ThemeSharedModule.forRoot({
  confirmationIcons: {
    info: 'fa fa-info-circle',
    success: 'fa fa-check-circle',
    warning: 'fa fa-exclamation-triangle',
    error: 'fa fa-times-circle',
    default: 'fa fa-question-circle',
  },
}),
...
```

```

```
```
```

```
## API
```

```
#### success
```

```
```js
```

```
success(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Confirmation.Options>,
) : Observable<Confirmation.Status>
```
```

> See the [`LocalizationParam`
type](<https://github.com/abpframework/abp/blob/master/npm/ng-packs/packages/core/src/lib/models/localization.ts#L6>) and [`Confirmation`
namespace](<https://github.com/abpframework/abp/blob/master/npm/ng-packs/packages/theme-shared/src/lib/models/confirmation.ts>)

```
#### warn
```

```
```js
```

```
warn(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Confirmation.Options>,
) : Observable<Confirmation.Status>
```
```

```
#### error
```

```
```js
```

```
error(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Confirmation.Options>,
) : Observable<Confirmation.Status>
```
```

```
#### info
```

```
```js
```

```
info(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Confirmation.Options>,
) : Observable<Confirmation.Status>
```
```

```
#### clear
```

```

```js
clear(
 status: Confirmation.Status = Confirmation.Status.dismiss
): void
```

```

- `status` parameter is the value of the confirmation closing event.

10.3.4.10 Loading Directive

Loading Directive

You may want to block a part of the UI and show a spinner for a while; the `LoadingDirective` directive makes this for you. `LoadingDirective` has been exposed by the `@abp/ng. theme. shared` package.

Getting Started

In order to use the `LoadingDirective` in an HTML template, the **`ThemeSharedModule`** should be imported into your module like this:

```

```js
// ...
import { ThemeSharedModule } from '@abp/ng. theme. shared';

@NgModule({
 //...
 imports: [..., ThemeSharedModule],
})
export class MyFeatureModule {}
```

```

Usage

The `LoadingDirective` is easy to use. The directive's selector is **`[abpLoading]`**. By adding the `abpLoading` attribute to an HTML element, you can activate the `LoadingDirective` for the HTML element when the value is true.

See an example usage:

```

```html
<div [abpLoading]="true">
 Lorem ipsum dolor sit, amet consectetur adipisicing elit. Laboriosam commodi quae
 aspernatur,
 corporis velit et suscipit id consequuntur amet minima expedita cum reiciendis dolorum

```

```

 cupiditate? Voluptas eaque voluptatum odio deleniti quo vel illum nemo accusamus nulla
ratione
 impedit dolorum expedita necessitatibus fugiat ullam beatae, optio eum cupiditate
ducimus
 architecto.
</div>
```

```

The `abpLoading` attribute has been added to the `<div>` element that contains very a long text inside to activate the `LoadingDirective`.

See the result:

```
![Loading directive result](./images/abp-loading.png)
```

10.3.4.11 Toast Overlay

Toast Overlay

You can use the `ToasterService` in @abp/ng. theme. shared package to display messages in an overlay by placing at the root level in your project.

Getting Started

You do not have to provide the `ToasterService` at module or component level, because it is already **provided in root****. You can inject and start using it immediately in your components, directives, or services.

```

```js
import { ToasterService } from '@abp/ng. theme. shared';

@Component({
 /* class metadata here */
})
class DemoComponent {
 constructor(private toaster: ToasterService) {}
}
```

```

Usage

You can use the `success`, `warn`, `error`, and `info` methods of `ToasterService` to display an overlay.

How to Display a Toast Overlay

```

```js
this.toaster.success("Message", "Title");
```

```

- The `ToasterService` methods accept three parameters that are `message`, `title`, and `options`.
- `success`, `warn`, `error`, and `info` methods return the id of opened toast overlay. The toast can be removed with this id.

How to Display a Toast Overlay With Given Options

Options can be passed as the third parameter to `success`, `warn`, `error`, and `info` methods:

```
```js
import { Toaster, ToasterService } from '@abp/ng.theme.shared';
//...

constructor(private toaster: ToasterService) {}

//...
const options: Partial<Toaster.ToastOptions> = {
 life: 10000,
 sticky: false,
 closable: true,
 tapToDismiss: true,
 messageLocalizationParams: ['Demo', '1'],
 titleLocalizationParams: []
};

this.toaster.error('AbpUi::EntityNotFoundError', 'AbpUi::Error', options);
```

```

- `life` option is the closing time in milliseconds. Default value is `5000`.
- `sticky` option keeps toast overlay on the screen by ignoring the `life` option when `true`. Default value is `false`.
- `closable` option displays the close icon on the toast overlay when it is `true`. Default value is `true`.
- `tapToDismiss` option, when `true`, allows closing the toast overlay by clicking over it. Default value is `false`.
- `yesText` is the text of the confirmation button. A localization key or localization object can be passed. Default value is `AbpUi::Yes`.
- `messageLocalizationParams` is the interpolation parameters for the localization of the message.
- `titleLocalizationParams` is the interpolation parameters for the localization of the title.

With the options above, the toast overlay looks like this:

```
![toast](./images/toast.png)
```

How to Remove a Toast Overlay

The open toast overlay can be removed manually via the `remove` method by passing the `id` of toast:

```

```js
const toastId = this.toaster.success("Message", "Title");

this.toaster.remove(toastId);
```

```

How to Remove All Toasts

The all open toasts can be removed manually via the `clear` method:

```

```js
this.toaster.clear();
```

```

Replacing ToasterService with 3rd party toaster libraries

If you want the ABP Framework to utilize 3rd party libraries for the toasters instead of the built-in one, you can provide a service that implements `Toaster.Service` interface, and provide it as follows (ngx-toastr library used in example):

- You can use *_LocalizationService_* for toaster messages translations.

```

```js
// your-custom-toaster.service.ts
import { Injectable } from '@angular/core';
import { Config, LocalizationService } from '@abp/ng.core';
import { Toaster } from '@abp/ng.theme.shared';
import { ToastrService } from 'ngx-toastr';

@Injectable()
export class CustomToasterService implements Toaster.Service {
 constructor(private toastr: ToastrService, private localizationService: LocalizationService) {}

 error(
 message: Config.LocalizationParam,
 title?: Config.LocalizationParam,
 options?: Partial<Toaster.ToastOptions>,
) {
 return this.show(message, title, 'error', options);
 }

 clear(): void {
 this.toastr.clear();
 }

 info(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam | undefined,
 options: Partial<Toaster.ToastOptions> | undefined,
): Toaster.ToasterId {

```

```

 return this.show(message, title, 'info', options);
 }

remove(id: number): void {
 this.toastr.remove(id);
}

show(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 severity: Toaster.Severity,
 options: Partial<Toaster.ToastOptions>,
): Toaster.ToasterId {
 const translatedMessage = this.localizationService.instant(message);
 const translatedTitle = this.localizationService.instant(title);
 const toasterOptions = {
 positionClass: 'toast-bottom-right',
 tapToDismiss: options.tapToDismiss,
 ... (options.sticky && {
 extendedTimeOut: 0,
 timeOut: 0,
 }),
 };
 const activeToast = this.toastr.show(
 translatedMessage,
 translatedTitle,
 toasterOptions,
 `toast-${severity}`,
);
 return activeToast.toastId;
}

success(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam | undefined,
 options: Partial<Toaster.ToastOptions> | undefined,
): Toaster.ToasterId {
 return this.show(message, title, 'success', options);
}

warn(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam | undefined,
 options: Partial<Toaster.ToastOptions> | undefined,
): Toaster.ToasterId {
 return this.show(message, title, 'warning', options);
}
}

```
```js
// app.module.ts

```

```

import { ToasterService } from '@abp/ng.theme.shared';

@NgModule({
 providers: [
 // ...
 {
 provide: ToasterService,
 useClass: CustomToasterService,
 },
]
})
```
## API

#### success

```js
success(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Toaster.ToastOptions>,
): number
```

- `Config` namespace can be imported from `@abp/ng.core`.
- `Toaster` namespace can be imported from `@abp/ng.theme.shared`.

> See the [LocalizationParam type](https://github.com/abpframework/abp/blob/master/npm/ng-packs/packages/core/src/lib/models/localization.ts#L6) and [Toaster namespace](https://github.com/abpframework/abp/blob/master/npm/ng-packs/packages/theme-shared/src/lib/models/toaster.ts)

#### warn

```js
warn(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Toaster.ToastOptions>,
): number
```

#### error

```js
error(
 message: Config.LocalizationParam,
 title: Config.LocalizationParam,
 options?: Partial<Toaster.ToastOptions>,
)
```

```

```

): number
```

```

### info

```

```js
info(
  message: Config.LocalizationParam,
  title: Config.LocalizationParam,
  options?: Partial<Toaster.ToastOptions>,
): number
```

```

### remove

```

```js
remove(id: number): void
```

```

Removes an open toast by the given id.

### clear

```

```js
clear(): void
```

```

Removes all open toasts.

## See Also

- [Confirmation Popup](./Confirmation-Service.md)

#### 10.3.4.12 Page Alerts

# Page Alerts

A page alert is useful for displaying an important message to the user. The ABP Framework provides an easy way to show the following alert to the user.

![angular-page-alert-example](./images/page-alert-warning-example.png)

You can simply import `PageAlertService` from `@abp/ng.theme.shared` and utilize it as follows:

```

```js
import { PageAlertService } from '@abp/ng.theme.shared';

@Component({
  ...
})

```

```

})
export class MyComponent {
  constructor(private service: PageAlertService) {}

  showWarning() {
    this.service.show({
      type: 'warning',
      message:
        'We will have a service interruption between 02:00 AM and 04:00 AM at October 23, 2023!',
      title: 'Service Interruption',
    });
  }
}
```

```

## `SHOW`

The method `show` accepts a single object that is type of `PageAlert`

```

```js
export interface PageAlert {
  type: 'primary' | 'secondary' | 'success' | 'danger' | 'warning' | 'info' | 'light' | 'dark';
  message: string;
  title?: string;
  dismissible?: boolean;
  messageLocalizationParams?: string[];
  titleLocalizationParams?: string[];
}
```

```

- \* `type` (Required): Defines what type of alert will be shown
- \* `message` (Required): The message who will be shown, also works with localization as well.
- \* `title` (Optional): The title of the message. If it is not provided, the title will be hidden.
- \* `dismissible` (Optional): Default is `true`. If enabled, a button on the top right corner will be shown to the users so that they can dismiss the message.
- \* `messageLocalizationParams` and `titleLocalizationParams` (Optional): If the message and/or the title is a key for localization service and contains some parameters, these fields could be used to pass those parameters.

#### An example with Localization

```

```typescript
this.service.show({
  type: 'danger',
  message: 'AbpAccount::PagerInfo{0}{1}{2}',
  messageLocalizationParams: ['10', '20', '30'],
  title: 'AbpAccount::EntityNotFoundErrorMessage',
  titleLocalizationParams: ['Test', 'id123'],
}
```

```

```
});
```

![angular-page-alert-with-params-example](./images/page-alert-with-params-example.png)

## ## Render HTML content

We can pass html content to the `title` and `message` parameters

- \* It allows static message or localization key
- \* [abpSafeHtml](https://github.com/abpframework/abp/blob/37b59a7f05202264505d002397dbb27d275740e1/npm/ng-packs/packages/core/src/lib/pipes/safe-html.pipe.ts#L6) pipe will sanitize html values

```
```typescript
this.service.show({
  type: 'success',
  title: `New <i><u>blog</u></i> published`,
  message: 'AbpApp::HtmlMessageWithParams{0}',
  messageLocalizationParams: ['admin'],
});
```

```

![angular-page-alert-with-html-example](./images/page-alert-with-html-example.png)

### 10.3.4.13 Ellipsis

#### # Ellipsis

Text inside an HTML element can be truncated easily with an ellipsis by using CSS. To make this even easier, you can use the `EllipsisDirective` which has been exposed by the `@abp/ng.theme.shared` package.

#### ## Getting Started

In order to use the `EllipsisDirective` in an HTML template, the \*\*`ThemeSharedModule`\*\* should be imported into your module like this:

```
```js
// ...
import { ThemeSharedModule } from '@abp/ng.theme.shared';

@NgModule({
  //...
  imports: [..., ThemeSharedModule],
})
export class MyFeatureModule {}
```

```
```
```

or **if you would not like to import** the `ThemeSharedModule`, you can import the **EllipsisModule** as shown below:

```
```js
// ...
import { EllipsisModule } from '@abp/ng.theme.shared';

@NgModule({
  //...
  imports: [..., EllipsisModule],
})
export class MyFeatureModule {}
```

Usage

The `EllipsisDirective` is very easy to use. The directive's selector is **abpEllipsis**. By adding the `abpEllipsis` attribute to an HTML element, you can activate the `EllipsisDirective` for the HTML element.

See an example usage:

```
```html
<p abpEllipsis>
 Lorem ipsum dolor sit, amet consectetur adipisicing elit. Laboriosam commodi quae
 aspernatur,
 corporis velit et suscipit id consequuntur amet minima expedita cum reiciendis dolorum
 cupiditate? Voluptas eaque voluptatum odio deleniti quo vel illum nemo accusamus nulla
 ratione
 impedit dolorum expedita necessitatibus fugiat ullam beatae, optio eum cupiditate
 ducimus
 architecto.
</p>
```

The `abpEllipsis` attribute has been added to the ``<p>`` element that containing very long text inside to activate the `EllipsisDirective`.

See the result:

`![Ellipsis directive result](./images/ellipsis-directive-result1.jpg)`

The long text has been truncated by using the directive.

The UI before using the directive looks like this:

`![Before using the EllipsisDirective](./images/ellipsis-directive-before.jpg)`

### Specifying Max Width of an HTML Element

An HTML element max width can be specified as shown below:

```
```html
<div [abpEllipsis]="" 100px' ">
  Lorem ipsum dolor sit amet consectetur adipisicing elit. Cumque, optio!
</div>

<div [abpEllipsis]="" 15vw' ">
  Lorem ipsum dolor sit amet consectetur adipisicing elit. Cumque, optio!
</div>

<div [abpEllipsis]="" 50%' ">
  Lorem ipsum dolor sit amet consectetur adipisicing elit. Cumque, optio!
</div>
```

```

See the result:

![Ellipsis directive result 2](./images/ellipsis-directive-result2.jpg)

#### 10.3.4.14 Context Strategy

```
ContextStrategy
```

`ContextStrategy` is an abstract class exposed by @abp/ng.core package. There are three context strategies extending it: `ComponentContextStrategy`, `TemplateContextStrategy`, and `NoContextStrategy`. Implementing the same methods and properties, all of these strategies help you define how projected content will get their context.

```
ComponentContextStrategy
```

`ComponentContextStrategy` is a class that extends `ContextStrategy`. It lets you **\*\*pass context to a projected component\*\***.

```
constructor
```

```
```js
constructor(public context: Partial<InferredInstanceOf<T>>) {}
```

```

- `T` refers to component type here, i.e. `Type<C>`.
- `InferredInstanceOf` is a utility type exposed by @abp/ng.core package. It infers component shape.
- `context` will be mapped to properties of the projected component.

```
setContext
```

```
```js
setContext(componentRef: ComponentRef<InferredInstanceOf<T>>):
Partial<InferredInstanceOf<T>>
```

```

This method maps each prop of the context to the component property with the same name and calls change detection. It returns the context after mapping.

#### ## TemplateContextStrategy

`TemplateContextStrategy` is a class that extends `ContextStrategy`. It lets you **\*\*pass context to a projected template\*\***.

##### ### constructor

```
```js
constructor(public context: Partial<InferredContextOf<T>>) {}
```

```

- `T` refers to template context type here, i.e. `TemplateRef<C>`.
- `InferredContextOf` is a utility type exposed by @abp/ng.core package. It infers context shape.
- `context` will be mapped to properties of the projected template.

##### ### setContext

```
```js
setContext(): Partial<InferredContextOf<T>>
```

```

This method does nothing and only returns the context, because template context is not mapped but passed in as parameter to `createEmbeddedView` method.

#### ## NoContextStrategy

`NoContextStrategy` is a class that extends `ContextStrategy`. It lets you **\*\*skip passing any context to projected content\*\***.

##### ### constructor

```
```js
constructor()
```

```

Unlike other context strategies, `NoContextStrategy` constructor takes no parameters.

#### #### setContext

```
```js
setContext(): undefined
````
```

Since there is no context, this method gets no parameters and will return `undefined`.

### ## Predefined Context Strategies

Predefined context strategies are accessible via `CONTEXT\_STRATEGY` constant.

#### #### None

```
```js
CONTEXT_STRATEGY.None()
````
```

This strategy will not pass any context to the projected content.

#### #### Component

```
```js
CONTEXT_STRATEGY.Component(context: Partial<InferredContextOf<T>>)
````
```

This strategy will help you pass the given context to the projected component.

#### #### Template

```
```js
CONTEXT_STRATEGY.Template(context: Partial<InferredContextOf<T>>)
````
```

This strategy will help you pass the given context to the projected template.

### ## See Also

- [ProjectionStrategy] (./Projection-Strategy.md)

#### 10.3.4.15 Cross Origin Strategy

## # CrossOriginStrategy

`CrossOriginStrategy` is a class exposed by @abp/ng.core package. Its instances define how a source referenced by an element will be retrieved by the browser and are consumed by other classes such as `LoadingStrategy` .

### ## API

#### ### constructor

```
```js
constructor(
  public crossorigin: 'anonymous' | 'use-credentials',
  public integrity?: string
)
````
```

- `crossorigin` is mapped to [the HTML attribute with the same name](<https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/crossorigin>).
- `integrity` is a hash for validating a remote resource. Its use is explained [[here](#)]([https://developer.mozilla.org/en-US/docs/Web/Security/Subresource\\_Integrity](https://developer.mozilla.org/en-US/docs/Web/Security/Subresource_Integrity)).

#### ### setCrossOrigin

```
```js
setCrossOrigin(element: HTMLElement): void
````
```

This method maps the aforementioned properties to the given `element` .

## ## Predefined Cross-Origin Strategies

Predefined cross-origin strategies are accessible via `CROSS\_ORIGIN\_STRATEGY` constant.

#### ### Anonymous

```
```js
CROSS_ORIGIN_STRATEGY.Anonymous(integrity?: string)
````
```

`crossorigin` will be set as `"[anonymous](#)"` and `integrity` is optional.

#### ### UseCredentials

```
```js
CROSS_ORIGIN_STRATEGY.UseCredentials(integrity?: string)
```

```

`crossorigin` will be set as `"`use-credentials`"` and `integrity` is optional.

#### 10.3.4.16 Dom Strategy

# DomStrategy

`DomStrategy` is a class exposed by `@abp/ng.core` package. Its instances define how an element will be attached to the DOM and are consumed by other classes such as `LoadingStrategy`.

## API

### constructor

```
```js
constructor(
  public target?: HTMLElement,
  public position?: InsertPosition
)
```

```

- `target` is an `HTMLElement` (`_default: document.head`).
- `position` defines where the created element will be placed. All possible values of `position` can be found [\[here\]](#) (<https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentElement>) (`_default: 'beforeend'`).

### insertElement

```
```js
insertElement(element: HTMLElement): void
```

```

This method inserts given `element` to `target` based on the `position`.

## Predefined Dom Strategies

Predefined dom strategies are accessible via `DOM\_STRATEGY` constant.

### AppendToBody

```
```js
DOM_STRATEGY.AppendToBody()
```
```

`insertElement` will place the given `element` at the end of `<body>`.

#### #### AppendToHead

```
```js
DOM_STRATEGY.AppendToHead()
```
```

`insertElement` will place the given `element` at the end of `<head>`.

#### #### PrependToHead

```
```js
DOM_STRATEGY.PrependToHead()
```
```

`insertElement` will place the given `element` at the beginning of `<head>`.

#### #### AfterElement

```
```js
DOM_STRATEGY.AfterElement(target: HTMLElement)
```
```

`insertElement` will place the given `element` after (as a sibling to) the `target`.

#### #### BeforeElement

```
```js
DOM_STRATEGY.BeforeElement(target: HTMLElement)
```
```

`insertElement` will place the given `element` before (as a sibling to) the `target`.

## ## See Also

- [DomInsertionService] (./Dom-Insertion-Service.md)
- [LazyLoadService] (./Lazy=Load-Service.md)
- [LoadingStrategy] (./Loading-Strategy.md)
- [ContentStrategy] (./Content-Strategy.md)
- [ProjectionStrategy] (./Projection-Strategy.md)

#### 10.3.4.17 Container Strategy

```
ContainerStrategy
```

`ContainerStrategy` is an abstract class exposed by @abp/ng.core package. There are two container strategies extending it: `ClearContainerStrategy` and `InsertIntoContainerStrategy`. Implementing the same methods and properties, both of these strategies help you define how your containers will be prepared and where your content will be projected.

```
API
```

`ClearContainerStrategy` is a class that extends `ContainerStrategy`. It lets you **\*\*clear a container before projecting content in it\*\***.

```
constructor
```

```
```js
constructor(
  public containerRef: ViewContainerRef,
  private index?: number, // works only in InsertIntoContainerStrategy
)
```

```

- `containerRef` is the `ViewContainerRef` that will be used when projecting the content.

```
getIndex
```

```
```js
getIndex(): number
```

```

This method return the given index clamped by `0` and `length` of the `containerRef`. For strategies without an index, it returns `0`.

```
prepare
```

```
```js
prepare(): void
```

```

This method is called before content projection. Based on used container strategy, it either clears the container or does nothing (noop).

```
ClearContainerStrategy
```

`ClearContainerStrategy` is a class that extends `ContainerStrategy`. It lets you **\*\*clear a container before projecting content in it\*\***.

```
InsertIntoContainerStrategy
```

`InsertIntoContainerStrategy` is a class that extends `ContainerStrategy`. It lets you **\*\*project your content at a specific node index in the container\*\***.

```
Predefined Container Strategies
```

Predefined container strategies are accessible via `CONTAINER\_STRATEGY` constant.

```
Clear
```

```
```js
CONTAINER_STRATEGY.Clear(containerRef: ViewContainerRef)
```
```

Clears given container before content projection.

```
Append
```

```
```js
CONTAINER_STRATEGY.Append(containerRef: ViewContainerRef)
```
```

Projected content will be appended to the container.

```
Prepend
```

```
```js
CONTAINER_STRATEGY.Prepend(containerRef: ViewContainerRef)
```
```

Projected content will be prepended to the container.

```
Insert
```

```
```js
CONTAINER_STRATEGY.Insert(
```
```

```
 containerRef: ViewContainerRef,
 index: number,
)
```

```

Projected content will be inserted into to the container at given index (clamped by `0` and `length` of the `containerRef`).

See Also

- [ProjectionStrategy] (./Projection-Strategy.md)

10.3.4.18 Content Security Strategy

ContentSecurityStrategy

`ContentSecurityStrategy` is an abstract class exposed by @abp/ng.core package. It helps you mark inline scripts or styles as safe in terms of [Content Security Policy](<https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy>).

API

constructor

```
```js
constructor(public nonce?: string)
```

```

- `nonce` enables whitelisting inline script or styles in order to avoid using `unsafe-inline` in [script-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/script-src#Unsafe_inline_script) and [style-src](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy/style-src#Unsafe_inline_styles) directives.

applyCSP

```
```js
applyCSP(element: HTMLScriptElement | HTMLStyleElement): void
```

```

This method maps the aforementioned properties to the given `element`.

LooseContentSecurityPolicy

`LooseContentSecurityPolicy` is a class that extends `ContentSecurityStrategy`. It requires `nonce` and marks given `

10.3.5 Customization

10.3.5.1 Customization Guide

Angular User Interface Customization Guide

- * [Replacing a component] (Component-Replacement.md)

10.3.5.2 Theming

10.3.5.2.1 Overall

Angular UI: Theming

Introduction

ABP Framework provides a complete **UI Theming** system with the following goals:

- * Reusable [application modules] (./../Modules/Index.md) are developed **theme-independent**, so they can work with any UI theme.
- * UI theme is **decided by the final application**.
- * The theme is distributed via an NPM package, so it is **easily upgradable**.
- * The final application can **customize** the selected theme.

In order to accomplish these goals, ABP Framework;

- * Determines a set of **base libraries** used and adapted by all the themes. So, module and application developers can depend on and use these libraries without depending on a particular theme.
- * Provides a system that consists of layout parts (like navigation menus and toolbars) that is implemented by all the themes. So, the modules and the application contribute to the layout to compose a consistent application UI.

Current Themes

Currently, three themes are **officially provided**:

- * The [Basic Theme] (Basic-Theme.md) is the minimalist theme with the plain Bootstrap style. It is **open source and free**.
- * The [Lepton Theme] (<https://commercial.abp.io/themes>) is a **commercial** theme developed by the core ABP team and is a part of the [ABP Commercial] (<https://commercial.abp.io/>) license.
- * The [LeptonX Theme] (<https://x.leptontheme.com/>) is a theme that has both [commercial] (<https://docs.abp.io/en/commercial/latest/themes/lepton-x/commercial/angular>) and [lite] (./../Themes/LeptonXLite/Angular.md) choices.

Overall

The Base Libraries

All the themes must depend on the [\[@abp/ng. theme. shared\]](https://www.npmjs.com/package/@abp/ng.theme.shared) NuGet package, so they are indirectly depending on the following libraries:

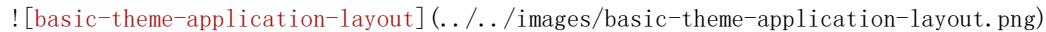
- * [\[Twitter Bootstrap\]](https://getbootstrap.com/) as the fundamental HTML/CSS framework.
- * [\[FontAwesome\]](https://fontawesome.com/) as the fundamental CSS font library.
- * [\[NG Bootstrap\]](https://ng-bootstrap.github.io/#/home) (<https://ng-bootstrap.github.io/#/home>) as a component library that supports the Bootstrap and adds extra components like modal and datepicker.
- * [\[Ngx-Datatable\]](https://swimlane.gitbook.io/ngx-datatable/) (<https://swimlane.gitbook.io/ngx-datatable/>) as a datatable library.
- * [\[ngx-validate\]](https://github.com/ng-turkey/ngx-validate) (<https://github.com/ng-turkey/ngx-validate>) a dynamic validation of reactive forms library.
- * [\[Chart.js\]](https://www.chartjs.org/) (<https://www.chartjs.org/>) as a widget library.

These libraries are selected as the base libraries and available to the applications and modules.

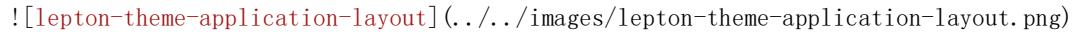
➤ Bootstrap's JavaScript part is not used since the NG Bootstrap library already provides the necessary functionalities to the Bootstrap components in a native way.

The Layout

All themes must define a layout for the application. The following image shows the user management page in the [\[Basic Theme\]](#) (Basic-Theme.md) application layout:

![basic-theme-application-layout] (./../images/basic-theme-application-layout.png)

And the same page is shown below with the [\[Lepton Theme\]](#) (<https://commercial.abp.io/themes>) application layout:

![lepton-theme-application-layout] (./../images/lepton-theme-application-layout.png)

As you can see, the page is the same, but the look is completely different in the themes above.

The application layout typically includes the following parts;

- * Main menu
- * Nav items area with the following components;
 - * User menu
 - * Language switch dropdown
- * [\[Page alerts\]](#) (Page-Alerts.md)
- * The page content (aka `<router-outlet>`)

Implementing a Theme

A theme is simply an NPM package and comes with startup templates.

The Easy Way

The easiest way to create a new theme is to add Basic Theme Source Code to your project via [\[ABP CLI\]](#) (./../CLI.md) command and customize it.

You can run the following command in **Angular** project directory to copy the source code to your solution:

```
`abp add-package @abp/ng.theme.basic --with-source-code`
```

Global/Component Styles

Angular can bundle global style files and component styles with components. See the [\[component styles\]](https://angular.io/guide/component-styles) (<https://angular.io/guide/component-styles>) guide on Angular documentation for more information.

Layout Parts

A typical layout consists of several parts. The theme should include the necessary parts in each layout.

****Example: The Basic Theme has the following parts for the Application Layout****

![basic-theme-application-layout-parts] (images/basic-theme-application-layout-parts.png)

The application code and the modules can only show contents in the Page Content part. If they need to change the other parts (to add a menu item, to add a nav item, to change the application name in the logo area...) they should use the ABP Framework APIs.

The following sections explain the fundamental parts pre-defined by the ABP Framework and can be implemented by the themes.

> It is a good practice to split the layout into components/partials, so the final application can override them partially for customization purpose.

Logo

The `application` object of an environment file should be configured to get the name and the logo URL of the application to render in the logo part. Additionally, `LogoComponent` can be replaced. See [\[Component Replacement\]](#) (Component-Replacement.md) document for more.

The [\[Application Startup Template\]](#) (./../Startup-Templates/Application.md) has an implementation of this interface to set the values by the application developer.

Main Menu / Routes

`RoutesService` service is used to manage the main menu items and render them on the layout.

****Example: Adding a route to the main menu****

```
```ts
import { RoutesService, eLayoutType } from '@abp/ng.core';
import { Component } from '@angular/core';

@Component({/* component metadata */})
export class AppComponent {
```

```

constructor(routes: RoutesService) {
 routes.add([
 {
 path: '/your-path',
 name: 'Your navigation',
 order: 101,
 iconClass: 'fas fa-question-circle',
 requiredPolicy: 'permission key here',
 layout: eLayoutType.application,
 },
 {
 path: '/your-path/child',
 name: 'Your child navigation',
 parentName: 'Your navigation',
 order: 1,
 requiredPolicy: 'permission key here',
 },
]);
}
```

```

See the [Modifying the Menu](Modifying-the-Menu.md) document to learn more about the navigation system.

Toolbar / Nav Items

`NavItemsService` service is used to get the menu's right part items and render on the layout. You can add an HTML content or Angular component as an element to render.

****Example: Adding an element to right part of the menu****

```

```ts
import { NavItemsService } from '@abp/ng.theme.shared';
import { Component } from '@angular/core';

@Component({
 template: `
 <input type="search" placeholder="Search" class="bg-transparent border-0 color-white" />
 `,
})
export class MySearchInputComponent {}

@Component(/* component metadata */)
export class AppComponent {
 constructor(private navItems: NavItemsService) {
 navItems.addItem([
 {
 id: 'MySearchInput',
 order: 1,
 }
]);
 }
}
```

```

```

        component: MySearchInputComponent,
    },
    {
        id: 'SignOutIcon',
        html: '<i class="fas fa-sign-out-alt fa-lg text-white m-2"></i>',
        action: () => console.log('Clicked the sign out icon'),
        order: 101, // puts as last element
    },
]);
}
```

```

› See the [How to Add an Element to Right Part of the Menu] (Modifying-the-Menu#how-to-add-an-element-to-right-part-of-the-menu) document to learn more on the nav items system.

The theme has a responsibility to add two pre-defined items to the toolbar: Language Selection and User Menu.

#### ##### Language Selection

Language Selection toolbar item is generally a dropdown that is used to switch between languages. `ConfigStateService` is used to get the list of available languages and `SessionStateService` is used to learn the current language.

`SessionStateService` is used to get and set the current language.

#### **\*\*Example: Get the currently selected language\*\***

```

```ts
import {SessionStateService} from '@abp/ng.core';

//...

constructor(private sessionState: SessionStateService) {
    const lang = this.sessionState.getLanguage()
}
```

```

#### **\*\*Example: Set the selected language\*\***

```

```ts
import {SessionStateService} from '@abp/ng.core';

//...

constructor(private sessionState: SessionStateService) {
    const lang = this.sessionState.setLanguage('en')
}
```

```

## ##### User Menu

`UserMenuService` is used to get the user menu's items and render on the dropdown. You can add/update/remove an item by using the service.

You can either pass a `component`, a piece of `html` or a `textTemplate` to render an item.

All of the options are shown below. You can choose either of them.

### **\*\*Example: Adding/updating/removing an user menu item\*\***

```
```ts
import { eUserMenuItems } from '@abp/ng.theme.basic';
import { UserMenuService } from '@abp/ng.theme.shared';
import { Component } from '@angular/core';
import { Router } from '@angular/router';

// make sure that you import this component in a NgModule
@Component({
  selector: 'abp-current-user-test',
  template: `
    <a class="dropdown-item pointer" (click)="data.action()">
      <i *ngIf="data.textTemplate.icon" [class]= "data.textTemplate.icon"></i>
      {{ data.textTemplate.text | abpLocalization }}
    </a>
  `,
})
export class UserMenuItemComponent {
  // you can inject the data through `INJECTOR_PIPE_DATA_TOKEN` token
  constructor(@Inject(INJECTOR_PIPE_DATA_TOKEN) public data: UserMenu) {}
}

@Component({/* component metadata */})
export class AppComponent {
  constructor(private userMenu: UserMenuService, private router: Router) {
    this.userMenu.addItem([
      {
        id: 'UserMenu.MyAccount',
        order: 1,
        // HTML example
        html: `<a class="dropdown-item pointer">My account</a>`,
        // text template example
        textTemplate: {
          text: 'AbpAccount::MyAccount',
          icon: 'fa fa-cog',
        },
        // component example
        component: UserMenuItemComponent,
      }
    ]);
  }
}
```

```

        action: () => {
          this.router.navigateByUrl('/account/manage');
        },
      },
    );
  }

  this.userMenu.patchItem(eUserMenuItem.MyAccount, {
    html: `<a class="dropdown-item pointer">My profile</a>`,
  });

  this.userMenu.removeItem(eUserMenuItem.Logout);
}
}
```

```

In the example above, added a new user menu item labeled "Reports", updated the "My account" item HTML content, and removed the "Logout" item.

See the result:

![image](https://user-images.githubusercontent.com/34455572/148387770-5b5e25fb-f855-447c-8ead-c04c69b6d6f7.png)

#### #### Page Alerts

`PageAlertService` service is used to get the current page alerts to render on the layout. See the [Page Alerts](Page-Alerts.md) document to learn more.

#### 10.3.5.2.2 Configuration

##### # Angular UI: Theme Configurations

Theme packages no longer import styles as CSS modules as of ABP version 6.0. The following style settings need to be included to the angular.json file in order for theme packages to load styles.

```

Lepton X Lite
```json
{
  "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/bootstrap-dim.css",
  "inject": false,
  "bundleName": "bootstrap-dim"
},
{
  "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/ng-bundle.css",
  "inject": false,
  "bundleName": "ng-bundle"
},
{
  "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/side-menu/layout-
bundle.css",

```

```

    "inject": false,
    "bundleName": "layout-bundle"
},
{
    "input": "node_modules/@abp/ng.theme.lepton-x/assets/css/abp-bundle.css",
    "inject": false,
    "bundleName": "abp-bundle"
},
{
    "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/bootstrap-dim rtl.css",
    "inject": false,
    "bundleName": "bootstrap-dim rtl"
},
{
    "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/ng-bundle.rtl.css",
    "inject": false,
    "bundleName": "ng-bundle.rtl"
},
{
    "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/side-menu/layout-
bundle.rtl.css",
    "inject": false,
    "bundleName": "layout-bundle.rtl"
},
{
    "input": "node_modules/@abp/ng.theme.lepton-x/assets/css/abp-bundle.rtl.css",
    "inject": false,
    "bundleName": "abp-bundle.rtl"
},
{
    "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/font-bundle.rtl.css",
    "inject": false,
    "bundleName": "font-bundle.rtl"
},
{
    "input": "node_modules/@volo/ngx-lepton-x-lite/assets/css/font-bundle.css",
    "inject": false,
    "bundleName": "font-bundle"
},
```
Theme Basic
```json
{
    "input": "node_modules/bootstrap/dist/css/bootstrap.rtl.min.css",
    "inject": false,
    "bundleName": "bootstrap-rtl.min"
},
{
    "input": "node_modules/bootstrap/dist/css/bootstrap.min.css",
    "inject": true,
    "bundleName": "bootstrap-ltr.min"
}

```

```
},
```
Lepton X [commercial] (https://docs.abp.io/en/commercial)
```json
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/dark.css",
  "inject": false,
  "bundleName": "dark"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/light.css",
  "inject": false,
  "bundleName": "light"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/dim.css",
  "inject": false,
  "bundleName": "dim"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/bootstrap-dim.css",
  "inject": false,
  "bundleName": "bootstrap-dim"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/bootstrap-dark.css",
  "inject": false,
  "bundleName": "bootstrap-dark"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/bootstrap-light.css",
  "inject": false,
  "bundleName": "bootstrap-light"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/ng-bundle.css",
  "inject": false,
  "bundleName": "ng-bundle"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/side-menu/layout-bundle.css",
  "inject": false,
  "bundleName": "layout-bundle"
},
{
  "input": "node_modules/@volosoft/abp.ng.theme.lepton-x/assets/css/abp-bundle.css",
  "inject": false,
  "bundleName": "abp-bundle"
},
{
  "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/dark rtl.css",
```

```
        "inject": false,
        "bundleName": "dark rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/light rtl.css",
        "inject": false,
        "bundleName": "light rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/dim rtl.css",
        "inject": false,
        "bundleName": "dim rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/bootstrap dim rtl.css",
        "inject": false,
        "bundleName": "bootstrap dim rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/bootstrap dark rtl.css",
        "inject": false,
        "bundleName": "bootstrap dark rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/bootstrap light rtl.css",
        "inject": false,
        "bundleName": "bootstrap light rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/ng bundle rtl.css",
        "inject": false,
        "bundleName": "ng bundle rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/side menu layout bundle rtl.css",
        "inject": false,
        "bundleName": "layout bundle rtl"
    },
    {
        "input": "node_modules/@volosoft/abp ng theme lepton-x/assets/css/abp bundle rtl.css",
        "inject": false,
        "bundleName": "abp bundle rtl"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/font bundle.css",
        "inject": false,
        "bundleName": "font bundle"
    },
    {
        "input": "node_modules/@volosoft/ngx-lepton-x/assets/css/font bundle rtl.css",
        "inject": false,
```

```
        "bundleName": "font-bundle.rtl"
    },
```
Theme Lepton [commercial] (https://docs.abp.io/en/commercial)
```json
{
    "input": "node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton1.min.css",
    "inject": false,
    "bundleName": "lepton1"
},
{
    "input": "node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton2.min.css",
    "inject": false,
    "bundleName": "lepton2"
},
{
    "input": "node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton3.min.css",
    "inject": false,
    "bundleName": "lepton3"
},
{
    "input": "node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton4.min.css",
    "inject": false,
    "bundleName": "lepton4"
},
{
    "input": "node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton5.min.css",
    "inject": false,
    "bundleName": "lepton5"
},
{
    "input": "node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton6.min.css",
    "inject": false,
    "bundleName": "lepton6"
},
{
    "input":
"node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton1 rtl.min.css",
    "inject": false,
    "bundleName": "lepton1 rtl"
},
{
    "input":
"node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton2 rtl.min.css",
    "inject": false,
    "bundleName": "lepton2 rtl"
},
{
    "input":
"node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton3 rtl.min.css",
    "inject": false,
```

```

    "inject": false,
    "bundleName": "lepton3.rtl"
},
{
  "input":
"node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton4.rtl.min.css",
  "inject": false,
  "bundleName": "lepton4.rtl"
},
{
  "input":
"node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton5.rtl.min.css",
  "inject": false,
  "bundleName": "lepton5.rtl"
},
{
  "input":
"node_modules/@volo/abp.ng.theme.lepton/dist/global/styles/lepton6.rtl.min.css",
  "inject": false,
  "bundleName": "lepton6.rtl"
},
```

```

#### 10.3.5.2.3 The Basic Theme

##### # Angular UI: Basic Theme

The Basic Theme is a theme implementation for the Angular UI. It is a minimalist theme that doesn't add any styling on top of the plain [Bootstrap](<https://getbootstrap.com/>). You can take the Basic Theme as the **\*\*base theme\*\*** and build your own theme or styling on top of it. See the *\*Customization\** section.

- If you are looking for a professional, enterprise ready theme, you can check the [Lepton Theme](<https://commercial.abp.io/themes>), which is a part of the [ABP Commercial](<https://commercial.abp.io/>).
- See the [Theming document](Theming.md) to learn about themes.

##### ## Installation

If you need to manually this theme, follow the steps below:

- \* Install the [`@abp/ng.theme.basic`](<https://www.npmjs.com/package/@abp/ng.theme.basic>) NPM package to your Angular project.
- \* Open the `src/app/app.module.ts` file, import `ThemeBasicModule` (it can be imported from `@abp/ng.theme.basic` package), and add `ThemeBasicModule.forRoot()` to the `imports` array.

\* Open the `src/app/shared/shared.module` file, import `ThemeBasicModule` (it can be imported from `@abp/ng.theme.basic` package), and add `ThemeBasicModule` to the `imports` and `exports` array.

The `ThemeBasicModule` is registered own layouts (`ApplicationLayoutComponent`, `AccountLayoutComponent`, `EmptyLayoutComponent`) to a service which is exposed by `@abp/ng.core` package on application initialization.

## ## Application Layout

![basic-theme-application-layout](../../images/basic-theme-application-layout.png)

Application Layout implements the following parts, in addition to the common parts mentioned above;

- \* Logo area
- \* Routes area
- \* Language selection & user menu
- \* [Page Alerts] (Page-Alerts.md)

See Application Layout components:

![application layout components](./images/layout-components.png)

## ### How to Use a Layout

Routes should be added to the menu by calling `add` method `RoutesService`. A layout can be set in the object of your route. See the [modifying the menu] (Modifying-the-Menu#how-to-add-a-navigation-element) for more information.

## ## Customization

You have two options two customize this theme:

### ### Overriding Styles / Components

In this approach, you continue to use the theme as an NPM package and customize the parts you need to. There are several ways to customize it;

#### #### Override the Styles

You can simply override the styles in the global styles (`src/styles.scss`) file of your application.

#### #### Override the Components

See the [Component Replacement] (Component-Replacement.md) to learn how you can replace components, customize and extend the user interface.

#### ### Copy & Customize

You can run the following [ABP CLI](../../CLI.md) command in **Angular** project directory to copy the source code to your solution:

```
`abp add-package @abp/ng.theme.basic --with-source-code`
```

----

Or, you can download the [source code](https://github.com/abpframework/abp/blob/dev/npm/ng-packs/packages/theme-basic) of the Basic Theme, manually copy the project content into your project (`projects/theme-basic` folder), open `angular.json` file and add configuration below to the `projects` object:

```
```json
{
  "projects": {
    ...
    "theme-basic": {
      "projectType": "library",
      "root": "projects/theme-basic",
      "sourceRoot": "projects/theme-basic/src",
      "prefix": "abp",
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:ng-packagr",
          "options": {
            "tsConfig": "projects/theme-basic/tsconfig.lib.json",
            "project": "projects/theme-basic/ng-package.json"
          },
          "configurations": {
            "production": {
              "tsConfig": "projects/theme-basic/tsconfig.lib.prod.json"
            }
          }
        }
      }
    }
  }
}```
```

Then, open the `tsconfig.json` file and add new paths as follows:

```
```json
"paths": {
 ...
 "@abp/ng.theme.basic": ["projects/theme-basic/src/public-api.ts"],
 "@abp/ng.theme.basic/testing": ["projects/theme-basic/testing/src/public-api.ts"]
}
```
```

You can now freely customize the theme based on your application requirements.

See Also

- * [Theming] (Theming.md)

10.3.5.2.4 LeptonX Lite

LeptonX Lite Angular UI

LeptonX Lite has implementation for the ABP Framework Angular Client. It's a simplified variation of the [LeptonX Theme] (<https://x.leptontheme.com/>).

- > If you are looking for a professional, enterprise ready theme, you can check the [LeptonX Theme] (<https://x.leptontheme.com/>), which is a part of [ABP Commercial] (<https://commercial.abp.io/>).
- > See the [Theming document] (<https://docs.abp.io/en/abp/latest/UI/AspNetCore/Theming>) to learn about themes.

Installation

This theme is ****already installed**** when you create a new solution using the startup templates. If you are using any other template, you can install this theme by following the steps below:

To add `LeptonX-lite` into your project,

- Install `@abp/ng.theme.lepton-x`

```
```bash
yarn add @abp/ng.theme.lepton-x
```
```

- Install `bootstrap-icons`

```
```bash
yarn add bootstrap-icons
```
```

- Then, we need to edit the styles array in `angular.json` to replace the existing style with the new one in the following link :

* [Styles – Angular UI] (../../UI/Angular/Theme-Configurations.md)

Note: You should remove the old theme styles from "angular.json" if you are switching from "ThemeBasic" or "Lepton."

Look at the [Theme Configurations] (../../UI/Angular/Theme-Configurations) list of styles. Depending on your theme, you can alter your styles in angular.json.

- Finally, remove `ThemeBasicModule` from `app.module.ts`, and import the related modules in `app.module.ts`

```
```js
import { ThemeLeptonXModule } from "@abp/ng.theme.lepton-x";
import { SideMenuLayoutModule } from "@abp/ng.theme.lepton-x/layouts";

@NgModule({
 imports: [
 // ...
 // do not forget to remove ThemeBasicModule or other old theme module
 // ThemeBasicModule.forRoot(),
 ThemeLeptonXModule.forRoot(),
 SideMenuLayoutModule.forRoot(),
],
 // ...
})
export class AppModule {}
````
```

Note: If you employ [Resource Owner Password Flow](<https://docs.abp.io/en/abp/latest/UI/Angular/Authorization#resource-owner-password-flow>) for authorization, you should import the following module as well:

```
```js
import { AccountLayoutModule } from "@abp/ng.theme.lepton-x/account";

@NgModule({
 // ...
 imports: [
 // ...
 AccountLayoutModule.forRoot(),
 // ...
],
 // ...
})
export class AppModule {}
````
```

To change the logos and brand color of `LeptonX`, simply add the following CSS to the `styles.scss`

```
```css
:root {
 --lpx-logo: url("/assets/images/logo.png");
 --lpx-logo-icon: url("/assets/images/logo-icon.png");
 --lpx-brand: #edae53;
}
````
```

- `--lpx-logo` is used to place the logo in the menu.

- `--lpx-logo-icon` is a square icon used when the menu is collapsed.
- `--lpx-brand` is a color used throughout the application, especially on active elements.

Server Side

In order to migrate to LeptonX on your server side projects (Host and/or AuthServer projects), please follow the [Server Side Migration](AspNetCore.md) document.

Customization

Layouts

The Angular version of LeptonX Lite provides **layout components** for your **user interface** on [ABP Framework Theming](<https://docs.abp.io/en/abp/latest/UI/Angular/Theming>). You can use the layouts to **organize your user interface**. You can replace the **layout components** and some parts of the **layout components** with the [ABP replaceable component system](<https://docs.abp.io/en/abp/latest/UI/Angular/Component-Replacement>).

The main responsibility of a theme is to **provide** the layouts. There are **three pre-defined layouts that must be implemented by all the themes:**

- **ApplicationLayoutComponent:** The **default** layout which is used by the **main** application pages.
- **AccountLayoutComponent:** Mostly used by the **account module** for **login**, **register**, **forgot password**... pages.
- **EmptyLayoutComponent:** The **Minimal** layout that **has no layout components** at all.

The **Layout components** and all the replacable components are predefined in `eThemeLeptonXComponents` as enum.

How to replace a component

```
```js
import { ReplaceableComponentsService } from '@abp/ng.core'; // imported
ReplaceableComponentsService
import { eIdentityComponents } from '@abp/ng.identity'; // imported eIdentityComponents
enum
import { eThemeLeptonXComponents } from '@abp/ng.theme.lepton-x'; // imported
eThemeLeptonXComponents enum

//...

@Component(/* component metadata */)
export class AppComponent {
 constructor(
 private replaceableComponents: ReplaceableComponentsService, // injected the service
) {
 this.replaceableComponents.add({
 component: YourNewApplicationLayoutComponent,
 key: eThemeLeptonXComponents.ApplicationLayout,
```

```
 });
}
```

```

See the [Component Replacement] (<https://docs.abp.io/en/abp/latest/UI/Angular/Component-Replacement>) documentation for more information on how to replace components.

Brand Component

The **brand component** is a simple component that can be used to display your brand. It contains a **logo** and a **company name**. You can change the logo via css but if you want to change logo component, the key is `eThemeLeptonXComponents.Logo`

```
```js
//...
this.replaceableComponents.add({
 component: YourNewLogoComponent,
 key: eThemeLeptonXComponents.Logo,
});
//...
```

```

![Brand component] (./../images/leptonxlite-brand-component.png)

Breadcrumb Component

On websites that have a lot of pages, **breadcrumb navigation** can greatly **enhance the way users find their way** around. In terms of **usability**, breadcrumbs reduce the number of actions a website **visitor** needs to take in order to get to a **higher-level page**, and they **improve** the **findability** of **website sections** and **pages**.

```
```js
//...
this.replaceableComponents.add({
 component: YourNewSidebarComponent,
 key: eThemeLeptonXComponents.Breadcrumb,
});
//...
```

```

![Breadcrumb component] (./../images/leptonxlite-breadcrumb-component.png)

Sidebar Menu Component

Sidebar menus have been used as a **directory for Related Pages** to a **Service** offering, **Navigation** items to a **specific service** or topic and even just as **Links** the user may be interested in.

```
```js
//...
this.replaceableComponents.add({
```

```
 component: YourNewSidebarComponent,
 key: eThemeLeptonXComponents.Sidebar,
 });
///...
```

```

Page Alerts Component

Provides contextual **feedback messages** for typical user actions with a handful of **available** and **flexible** **alert messages**. Alerts are available for any length of text, as well as an **optional dismiss button**.

```
![Page alerts component](../../images/leptonxlite-page-alerts-component.png)
```

```
```js
///...
this.replaceableComponents.add({
 component: YourNewPageAlertContainerComponent,
 key: eThemeLeptonXComponents.PageAlertContainer,
});
///...
```

```

Toolbar Component

```
![Breadcrumb component](../../images/leptonxlite-toolbar-component.png)
```

Toolbar items are used to add **extra functionality to the toolbar**. The toolbar is a **horizontal bar** that **contains** a group of **toolbar items**.

```
```js
///...
this.replaceableComponents.add({
 component: YourNewNavItemsComponent,
 key: eThemeLeptonXComponents.NavItems,
});
///...
```

```

Toolbar Items

There are two parts to the toolbar. The first is Language-Switch. The second is the User-Profile element. You can swap out each of these parts individually.

Language Switch Component

Think about a **multi-lingual** website and the first thing that could **hit your mind** is **the language switch component**. A **navigation bar** is a **great place** to **embed a language switch**. By embedding the language switch in the navigation bar of your

website, you would **make it simpler** for users to **find it** and **easily** switch the **language** **without trying to locate it across the website.**

![Language switch component](../../images/leptonxlite-language-switch-component.png)

```
```js
///
this.replaceableComponents.add({
 component: YourNewLanguagesComponent,
 key: eThemeLeptonXComponents.Languages,
});
///...
```

#### ## User Menu Component

The **User Menu** is the **menu** that **drops down** when you **click your name** or **profile picture** in the **upper right corner** of your page (**in the toolbar**). It drops down options such as **Settings**, **Logout**, etc.

![User menu component](../../images/leptonxlite-user-menu-component.png)

```
```js
///
this.replaceableComponents.add({
  component: YourNewCurrentUserComponent,
  key: eThemeLeptonXComponents.CurrentUser,
});
///...
```
```

Note: The language selection component in the Volo app is not replaceable. It is part of the settings menu.

#### ## Mobile Navbar Component

The **mobile navbar component** is used to display the **navbar menu on mobile devices**. The mobile navbar component is a **dropdown menu** that contains language selection and user menu.

![Mobile user menu component](../../images/leptonxlite-mobile-user-menu-component.png)

```
```js
///
this.replaceableComponents.add({
  component: YourNewMobileNavbarComponent,
  key: eThemeLeptonXComponents.MobileNavbar,
});
///...
```
```

#### ## Mobile Navbar Items.

There are two parts of the mobile navbar. The mobile navbar has Language-Switch and User-Profile. You can swap out each of these parts individually.

The Mobile language-Selection component key is `eThemeLeptonXComponents.MobileLanguageSelection`.

The Mobile User-Profile component key is `eThemeLeptonXComponents.MobileUserProfile`.

## ## Footer Component

![Angular Footer Component](../../../../images/angular-footer.png)

The Footer is the section of content at the very bottom of the site. This section of the content can be modified.

Inject **\*\*FooterLinksService\*\*** and use the **\*\*setFooterInfo\*\*** method of **\*\*FooterLinksService\*\***

to assign path or link and description.

**\*\*descUrl\*\*** and **\*\*footerLinks\*\*** are nullable. Constant **\*\*footerLinks\*\*** are on the right side of footer.

```
```js
///

const footerLinks = [
  {
    link: "/components/bootstrap/badge",
    text: "Manage Your Profile",
  },
  {
    link: "/components/bootstrap/border",
    text: "My Security Logs",
  },
];

const footerInfo: FooterNav = {
  desc: "Home",
  descUrl: "/components/home",
  footerLinks: footerLinks,
};

this.footerLinksService.setFooterInfo(footerInfo);

///...
```

If you want to change the footer component, the key is `eThemeLeptonXComponents.Footer`

```
```js
///

this.replaceableComponents.add({
 component: YourNewFooterComponent,
```

```
 key: eThemeLeptonXComponents.Footer,
});
///...
```

#### 10.3.5.3 Modifying the Menu

##### # Modifying the Menu

The menu is inside the `ApplicationLayoutComponent` in the `@abp/ng.theme.basic` package. There are several methods for modifying the menu elements. This document covers these methods. If you would like to replace the menu completely, please refer to [Component Replacement documentation] (./Component-Replacement.md) and learn how to replace a layout.

##### ## How to Add a Logo

The `logoUrl` property in the environment variables is the url of the logo.

You can add your logo to `src/assets` folder and set the `logoUrl` as shown below:

```
```js  
export const environment = {  
  // other configurations  
  application: {  
    name: 'MyProjectName',  
    logoUrl: 'assets/logo.png',  
  },  
  // other configurations  
};  
```
```

##### ## How to Add a Navigation Element

###### ### Via `RoutesService`

You can add routes to the menu by calling the `add` method of `RoutesService`. It is a singleton service, i.e. provided in root, so you can inject and use it immediately.

```
```js  
import { RoutesService, eLayoutType } from '@abp/ng.core';  
import { Component } from '@angular/core';  
  
@Component({* component metadata */}  
export class AppComponent {  
  constructor(routes: RoutesService) {  
    routes.add([  
      {
```

```

        path: '/your-path',
        name: 'Your navigation',
        order: 101,
        iconClass: 'fas fa-question-circle',
        requiredPolicy: 'permission key here',
        layout: eLayoutType.application,
    },
    {
        path: '/your-path/child',
        name: 'Your child navigation',
        parentName: 'Your navigation',
        order: 1,
        requiredPolicy: 'permission key here',
    },
]);
}
```

```

An alternative and probably cleaner way is to use a route provider. First create a provider:

```

```js
// route.provider.ts
import { RoutesService, eLayoutType } from '@abp/ng.core';
import { APP_INITIALIZER } from '@angular/core';

export const APP_ROUTE_PROVIDER = [
    { provide: APP_INITIALIZER, useFactory: configureRoutes, deps: [RoutesService], multi: true },
];

function configureRoutes(routes: RoutesService) {
    return () => {
        routes.add([
            {
                path: '/your-path',
                name: 'Your navigation',
                requiredPolicy: 'permission key here',
                order: 101,
                iconClass: 'fas fa-question-circle',
                layout: eLayoutType.application,
            },
            {
                path: '/your-path/child',
                name: 'Your child navigation',
                parentName: 'Your navigation',
                requiredPolicy: 'permission key here',
                order: 1,
            },
        ]);
    };
}
```

```

```
}
```

We can also define a group for navigation elements. It's an optional property  
- **\*\*Note:\*\*** It'll also include groups that were defined at the modules

```
```js
// route.provider.ts
import { RoutesService } from '@abp/ng.core';

function configureRoutes(routes: RoutesService) {
    return () => {
        routes.add([
            {
                //etc..
                group: 'ModuleName::GroupName'
            },
            {
                path: '/your-path/child',
                name: 'Your child navigation',
                parentName: 'Your navigation',
                requiredPolicy: 'permission key here',
                order: 1,
            },
        ]);
    };
}
```

```

To get the route items as grouped we can use the `groupedVisible` (or Observable one `groupedVisible\$`) getter methods

- It returns `RouteGroup<T>[]` if there is any group in the route tree, otherwise it returns `undefined`

```
```js
import { ABP, RoutesService, RouteGroup } from "@abp/ng.core";
import { Component } from "@angular/core";

@Component({/* component metadata */})
export class AppComponent {
    visible: RouteGroup<ABP.Route>[] | undefined = this.routes.groupedVisible;
    //Or
    visible$: Observable<RouteGroup<ABP.Route>[] | undefined> = this.routes.groupedVisible$;

    constructor(private routes: RoutesService) {}
}
```

```

...and then in app.module.ts...

- The `groupedVisible` method will return the `Others` group for ungrouped items, the default key is `AbpUi::OthersGroup`, we can change this `key` via the `OTHERS\_GROUP` injection token

```

```js
import { NgModule } from '@angular/core';
import { OTHERS_GROUP } from '@abp/ng.core';
import { APP_ROUTE_PROVIDER } from './route.provider';

@NgModule({
  providers: [
    APP_ROUTE_PROVIDER,
    {
      provide: OTHERS_GROUP,
      useValue: 'ModuleName::MyOthersGroupKey',
    },
  ],
  // imports, declarations, and bootstrap
})
export class AppModule {}
```

```

Here is what every property works as:

- `path` is the absolute path of the navigation element.
- `name` is the label of the navigation element. A localization key or a localization object can be passed.
- `parentName` is a reference to the `name` of the parent route in the menu and is used for creating multi-level menu items.
- `requiredPolicy` is the permission key to access the page. See the [Permission Management document](./Permission-Management.md)
- `order` is the order of the navigation element. "Administration" has an order of `100`, so keep that in mind when ordering top level menu items.
- `iconClass` is the class of the `i` tag, which is placed to the left of the navigation label.
- `layout` defines in which layout the route is loaded. (default: `eLayoutType.empty`)
- `invisible` makes the item invisible in the menu. (default: `false`)
- `group` is an optional property that is used to group together related routes in an application. (type: `string`, default: `AbpUi::OthersGroup`)

#### ### Via `routes` Property in `AppRoutingModule`

You can define your routes by adding `routes` as a child property to `data` property of a route configuration in the `app-routing.module`. The `@abp/ng.core` package organizes your routes and stores them in the `RoutesService`.

You can add the `routes` property like below:

```

```js
{
  path: 'your-path',
  data: {
    routes: {
      name: 'Your navigation',
      order: 101,
    }
  }
}
```

```

```

iconClass: 'fas fa-question-circle',
requiredPolicy: 'permission key here',
children: [
 {
 path: 'child',
 name: 'Your child navigation',
 order: 1,
 requiredPolicy: 'permission key here',
 },
],
},
},
},
```

```

Alternatively, you can do this:

```

```js
{
 path: 'your-path',
 data: {
 routes: [
 {
 path: '/your-path',
 name: 'Your navigation',
 order: 101,
 iconClass: 'fas fa-question-circle',
 requiredPolicy: 'permission key here',
 },
 {
 path: '/your-path/child',
 name: 'Your child navigation',
 parentName: 'Your navigation',
 order: 1,
 requiredPolicy: 'permission key here',
 },
] as ABP.Route[], // can be imported from @abp/ng.core
 },
}
```

```

The advantage of the second method is that you are not bound to the parent/child structure and use any paths you like.

After adding the `routes` property as described above, the navigation menu looks like this:

```
![navigation-menu-via-app-routing](./images/navigation-menu-via-app-routing.png)
```

How to Patch or Remove a Navigation Element

The ``patch`` method of ``RoutesService`` finds a route by its name and replaces its configuration with the new configuration passed as the second parameter. Similarly, ``remove`` method finds a route and removes it along with its children.

```
```js
// this.routes is instance of RoutesService
// eThemeSharedRouteNames enum can be imported from @abp/ng.theme.shared

const dashboardRouteConfig: ABP.Route = {
 path: '/dashboard',
 name: '::Menu:Dashboard',
 parentName: '::Menu:Home',
 order: 1,
 layout: eLayoutType.application,
};

const newHomeRouteConfig: Partial<ABP.Route> = {
 iconClass: 'fas fa-home',
 parentName: eThemeSharedRouteNames.Administration,
 order: 0,
};

this.routes.add([dashboardRouteConfig]);
this.routes.patch('::Menu:Home', newHomeRouteConfig);
this.routes.remove(['Your navigation']);
```

```

- Moved the `_Home_` navigation under the `_Administration_` dropdown based on given `'parentName'`.
- Added an icon to `_Home_`.
- Specified the order and made `_Home_` the first item in list.
- Added a route named `_Dashboard_` as a child of `_Home_`.
- Removed `_Your navigation_` along with its child route.

After the operations above, the new menu looks like below:

![navigation-menu-after-patching](./images/navigation-menu-after-patching.png)

How to Add an Element to Right Part of the Menu

You can add elements to the right part of the menu by calling the ``addItems`` method of ``NavItemsService``. It is a singleton service, i.e. provided in root, so you can inject and use it immediately.

```
```js
import { NavItemsService } from '@abp/ng.theme.shared';
import { Component } from '@angular/core';

@Component({
 template: `

```

```

 <input type="search" placeholder="Search" class="bg-transparent border-0 color-white"
 />
 ,
))
export class MySearchInputComponent {}

@Component(/* component metadata */)
export class AppComponent {
 constructor(private navItems: NavItemsService) {
 navItems.addItems([
 {
 id: 'MySearchInput',
 order: 1,
 component: MySearchInputComponent,
 },
 {
 id: 'SignOutIcon',
 html: '<i class="fas fa-sign-out-alt fa-lg text-white m-2"></i>',
 action: () => console.log('Clicked the sign out icon'),
 order: 101, // puts as last element
 },
]);
 }
}
```

```

This inserts a search input and a sign out icon to the menu. The final UI looks like below:

![:navigation-menu-search-input](./images/navigation-menu-search-input.png)

> The default elements have an order of `100`. If you want to place a custom element before the defaults, assign an order number up to `99`. If you want to place a custom element after the defaults, assign orders starting from `101`. Finally, if you must place an item between the defaults, patch the default element orders as described below. A warning though: We may add another default element in the future and it too will have an order number of `100`.

How to Patch or Remove an Right Part Element

The `patchItem` method of `NavItemsService` finds an element by its `id` property and replaces its configuration with the new configuration passed as the second parameter. Similarly, `removeItem` method finds an element and removes it.

```

```js
export class AppComponent {
 constructor(private navItems: NavItemsService) {
 navItems.patchItem(eThemeBasicComponents.Languages, {
 requiredPolicy: 'new policy here',
 order: 1,
 });
}
```

```

```

        navItems.removeItem(eThemeBasicComponents.CurrentUser);
    }
}
```
* Patched the languages dropdown element with new `requiredPolicy` and new `order`.
* Removed the current user dropdown element.

```

#### 10.3.5.4 Sorting Navigation Elements

##### # Sorting Navigation Elements

This documentation describes how the navigation elements are sorted and how to change this default behaviour.

When you want to add the `Navigation Element` you can use the `RoutesService`. For more details, see the [document](<https://docs.abp.io/en/abp/latest/UI/Angular/Modifying-the-Menu#how-to-add-a-navigation-element>).

However, in this documentation, we will talk more about how to sort the navigation elements with the `order` attribute from the `Routes Service`.

##### ### Order Property

- This parameter is optional and is used for sorting purposes.
- If you define this property it will be sorted by the default sorting function.
- You can edit this function.

##### \*\*Default Compare Function;\*\*

```

``compare-func.token.ts``
``ts
export const SORT_COMPARE_FUNC = new InjectionToken<0 | 1 | -1>('SORT_COMPARE_FUNC');

export function compareFuncFactory() {
 const localizationService = inject(LocalizationService);
 const fn = (a, b) => {
 const aName = localizationService.instant(a.name);
 const bName = localizationService.instant(b.name);

 const aNumber = a.order;
 const bNumber = b.order;

 if (!Number.isInteger(aNumber)) return 1;
 if (!Number.isInteger(bNumber)) return -1;

 if (aNumber > bNumber) return 1
 if (aNumber < bNumber) return -1

 if (aName > bName) return 1;
 if (aName < bName) return -1;
 }
}
``
```

```

 return 0
 }
 return fn;
}
```

```

****What does this function do?****

- if the order property is defined, then it will be sorted by the order value.
- if both of the navigation elements have the same order value then it will be sorted by the name.
- If the order property is not defined, it will be the last element and the unordered navs will be sorted by name.

You can edit this sorting function behaviour as you wish.

10.3.5.5 Component Replacement

Component Replacement

You can replace some ABP components with your custom components.

The reason that you ****can replace**** but ****cannot customize**** default ABP components is disabling or changing a part of that component can cause problems. So we named those components as *_Replaceable Components_*.

How to Replace a Component

Create a new component that you want to use instead of an ABP component. Add that component to `declarations` and `entryComponents` in the `AppModule`.

Then, open the `app.component.ts` and execute the `add` method of `ReplaceableComponentsService` to replace your component with an ABP component as shown below:

```

```js
import { ReplaceableComponentsService } from '@abp/ng.core'; // imported
ReplaceableComponentsService
import { eIdentityComponents } from '@abp/ng.identity'; // imported eIdentityComponents
enum
//...

@Component(/* component metadata */)
export class AppComponent {
 constructor(
 private replaceableComponents: ReplaceableComponentsService, // injected the service
) {
 this.replaceableComponents.add({
 component: YourNewRoleComponent,
 key: eIdentityComponents.Roles,
 });
 }
}
```

```

```
}
```

![Example Usage](./images/component-replacement.gif)

How to Replace a Layout

Each ABP theme module has 3 layouts named `ApplicationLayoutComponent`, `AccountLayoutComponent`, `EmptyLayoutComponent`. These layouts can be replaced the same way.

> A layout component template should contain `<router-outlet></router-outlet>` element.

The example below describes how to replace the `ApplicationLayoutComponent`:

Run the following command to generate a layout in `angular` folder:

```
```bash
yarn ng generate component my-application-layout
````
```

Add the following code in your layout template (`my-application-layout.component.html`) where you want the page to be loaded.

```
```html
<router-outlet></router-outlet>
````
```

Open `app.component.ts` in `src/app` folder and modify it as shown below:

```
```js
import { ReplaceableComponentsService } from '@abp/ng.core'; // imported
ReplaceableComponentsService
import { eThemeBasicComponents } from '@abp/ng.theme.basic'; // imported
eThemeBasicComponents enum for component keys
import { MyApplicationLayoutComponent } from './my-application-layout/my-application-
layout.component'; // imported MyApplicationLayoutComponent

@Component(/* component metadata */)
export class AppComponent {
 constructor(
 private replaceableComponents: ReplaceableComponentsService, // injected the service
) {
 this.replaceableComponents.add({
 component: MyApplicationLayoutComponent,
 key: eThemeBasicComponents.ApplicationLayout,
 });
 }
}
````
```

› If you like to replace a layout component at runtime (e.g: changing the layout by pressing a button), pass the second parameter of the `add` method of `ReplaceableComponentsService` as true. DynamicLayoutComponent loads content using a router-outlet. When the second parameter of the `add` method is true, the route will be refreshed, so use it with caution. Your component state will be gone and any initiation logic (including HTTP requests) will be repeated.

Layout Components

![Layout Components](./images/layout-components.png)

How to Replace LogoComponent

![LogoComponent](./images/logo-component.png)

Run the following command in `angular` folder to create a new component called `LogoComponent`.

```
```bash
yarn ng generate component logo --inlineTemplate --inlineStyle
````
```

Open the generated `logo.component.ts` in `src/app/logo` folder and replace its content with the following:

```
```js
import { Component } from '@angular/core';

@Component({
 selector: 'app-logo',
 template: `
 <a class="navbar-brand" routerLink="/"
 <!-- Change the img src -->

 `,
})
export class LogoComponent {}````
```

Open `app.component.ts` in `src/app` folder and modify it as shown below:

```
```js
import { ..., ReplaceableComponentsService } from '@abp/ng.core'; // imported
ReplaceableComponentsService
import { LogoComponent } from './logo/logo.component'; // imported LogoComponent````
```

```

import { eThemeBasicComponents } from '@abp/ng.theme.basic'; // imported
eThemeBasicComponents
//...

@Component(/* component metadata */)
export class AppComponent implements OnInit {
    constructor(..., private replaceableComponents: ReplaceableComponentsService) {} // injected ReplaceableComponentsService

    ngOnInit() {
        //...

        this.replaceableComponents.add({
            component: LogoComponent,
            key: eThemeBasicComponents.Logo,
        });
    }
}
```

```

The final UI looks like below:

![New logo](./images/replaced-logo-component.png)

#### #### How to Replace RoutesComponent

![RoutesComponent](./images/routes-component.png)

Run the following command in `angular` folder to create a new component called `RoutesComponent`.

```

```bash
yarn ng generate component routes
```

```

Open the generated `routes.component.ts` in `src/app/routes` folder and replace its content with the following:

```

```js
import { Component, HostBinding } from '@angular/core';

@Component({
    selector: 'app-routes',
    templateUrl: 'routes.component.html',
})
export class RoutesComponent {
    @HostBinding('class.mx-auto')
    marginAuto = true;

    get smallScreen() {
        return window.innerWidth < 992;
    }
}
```

```

```
}
```

Import the `SharedModule` to the `imports` array of `AppModule`:

```
```js
// app.module.ts

import { SharedModule } from './shared/shared.module';

@NgModule({
  imports: [
    //...
    SharedModule
  ]
})
```
```

```

Open the generated `routes.component.html` in `src/app/routes` folder and replace its content with the following:

```
```html
<ul class="navbar-nav">
 <li class="nav-item">

 <i class="fas fa-home"></i> {{{{':Menu:Home' | abpLocalization}}}}

 <li class="nav-item">
 <i class="fas fa-newspaper mr-1"></i>My
 Page

 <li
 #navbarRootDropdown
 [abpVisibility]="routeContainer"
 class="nav-item dropdown"
 display="static"
 (click)="
 navbarRootDropdown.expand
 ? (navbarRootDropdown.expand = false)
 : (navbarRootDropdown.expand = true)
 "
 >

 <i class="fas fa-wrench"></i>
 {{{'AbpUiNavigation::Menu:Administration' | abpLocalization}}}
 <div
 #routeContainer
 class="dropdown-menu border-0 shadow-sm"
 (click)="event.preventDefault(); event.stopPropagation()"
 [class.d-block]=""smallScreen && navbarRootDropdown.expand""
```

```

>
<div
 class="dropdown-submenu"
 ngbDropdown
 #dropdownSubmenu="ngbDropdown"
 placement="right-top"
 [autoClose]="true"
 *abpPermission="" AbpIdentity.Roles || AbpIdentity.Users"
>
 <div ngbDropdownToggle [class.dropdown-toggle]="false">
 <a
 abpEllipsis="210px"
 [abpEllipsisEnabled]!="smallScreen"
 role="button"
 class="btn d-block text-start dropdown-toggle"
 >
 <i class="fa fa-id-card-o"></i>
 {${{ 'AbpIdentity::Menu:IdentityManagement' | abpLocalization }}}

 </div>
 <div
 #childrenContainer
 class="dropdown-menu border-0 shadow-sm"
 [class.d-block]="$ctrl.isSmallScreen && dropdownSubmenu.isOpen()"
 >
 <div class="dropdown-submenu" *abpPermission="" AbpIdentity.Roles">

 {${{ 'AbpIdentity::Roles' | abpLocalization }}}

 </div>
 <div class="dropdown-submenu" *abpPermission="" AbpIdentity.Users">

 {${{ 'AbpIdentity::Users' | abpLocalization }}}

 </div>
 </div>
</div>

<div
 class="dropdown-submenu"
 ngbDropdown
 #dropdownSubmenu="ngbDropdown"
 placement="right-top"
 [autoClose]="true"
 *abpPermission="" AbpTenantManagement.Tenants"
>
 <div ngbDropdownToggle [class.dropdown-toggle]="false">
 <a
 abpEllipsis="210px"
 [abpEllipsisEnabled]!="smallScreen"
 role="button"
 class="btn d-block text-start dropdown-toggle"

```

```

 >
 <i class="fa fa-users"></i>
 {{'AbpTenantManagement::Menu:TenantManagement' | abpLocalization}}%

 </div>
 <div
 #childrenContainer
 class="dropdown-menu border-0 shadow-sm"
 [class.d-block]="smallScreen && dropdownSubmenu.isOpen()"
 >
 <div class="dropdown-submenu" *abpPermission="AbpTenantManagement.Tenants">

 {{'AbpTenantManagement::Tenants' | abpLocalization}}%

 </div>
 </div>
 </div>


```

```

Open `app.component.ts` in `src/app` folder and modify it as shown below:

```

```js
import { ..., ReplaceableComponentsService } from '@abp/ng.core'; // imported
ReplaceableComponentsService
import { RoutesComponent } from './routes/routes.component'; // imported RoutesComponent
import { eThemeBasicComponents } from '@abp/ng.theme.basic'; // imported
eThemeBasicComponents
//...

@Component(/* component metadata */)
export class AppComponent implements OnInit {
 constructor(..., private replaceableComponents: ReplaceableComponentsService) {} // injected ReplaceableComponentsService

 ngOnInit() {
 //...

 this.replaceableComponents.add({
 component: RoutesComponent,
 key: eThemeBasicComponents.Routes,
 });
 }
}
```

```

The final UI looks like below:

![New routes](./images/replaced-routes-component.png)

How to Replace NavItemsComponent

![NavItemsComponent](./images/nav-items-component.png)

Run the following command in `angular` folder to create a new component called `NavItemsComponent`.

```
```bash
yarn ng generate component nav-items
````
```

Open the generated `nav-items.component.ts` in `src/app/nav-items` folder and replace the content with the following:

```
```js
import {
 AuthService,
 ConfigStateService,
 CurrentUserDto,
 LanguageInfo,
 NAVIGATE_TO_MANAGE_PROFILE,
 SessionStateService,
} from '@abp/ng.core';
import { Component, Inject } from '@angular/core';
import { Observable } from 'rxjs';
import { map } from 'rxjs/operators';
import snq from 'snq';

@Component({
 selector: 'app-nav-items',
 templateUrl: 'nav-items.component.html',
})
export class NavItemsComponent {
 currentUser$: Observable<CurrentUserDto> = this.configState.getOne$('currentUser');
 selectedTenant$ = this.sessionState.getTenant$();

 languages$: Observable<LanguageInfo[]> =
this.configState.getDeep$('localization.languages');

 get smallScreen(): boolean {
 return window.innerWidth < 992;
 }

 get defaultLanguage$(): Observable<string> {
 return this.languages$.pipe(
 map(
 languages =>
 snq(
 () => languages.find(lang => lang.cultureName ===
this.selectedLangCulture).displayName
),
 ''
)
);
 }
}
```

```

)
);
}

get dropdownLanguages$(): Observable<LanguageInfo[]> {
 return this.languages$.pipe(
 map(
 languages =>
 snq(() => languages.filter(lang => lang.cultureName !==
this.selectedLangCulture)),
 []
)
);
}

get selectedLangCulture(): string {
 return this.sessionState.getLanguage();
}

constructor(
 @Inject(NAVIGATE_TO_MANAGE_PROFILE) public navigateToManageProfile,
 private configState: ConfigStateService,
 private authService: AuthService,
 private sessionState: SessionStateService
) {}

onChangeLang(cultureName: string) {
 this.sessionState.setLanguage(cultureName);
}

navigateToLogin() {
 this.authService.navigateToLogin();
}

logout() {
 this.authService.logout().subscribe();
}
}
```

```

Import the `SharedModule` to the `imports` array of `AppModule`:

```

```js
// app.module.ts

import { SharedModule } from './shared/shared.module';

@NgModule({
 imports: [
 //...
 SharedModule
]
})

```

```
)
```
```

Open the generated `nav-items.component.html` in `src/app/nav-items` folder and replace the content with the following:

```
```html  
<ul class="navbar-nav">
 <input type="search" placeholder="Search" class="bg-transparent border-0 text-white" />
 <li class="nav-item d-flex align-items-center">
 <div
 *ngIf="(dropdownLanguages$ | async)?.length > 0"
 class="dropdown"
 ngbDropdown
 #languageDropdown="ngbDropdown"
 display="static"
 >
 <a
 ngbDropdownToggle
 class="nav-link"
 href="javascript:void(0)"
 role="button"
 id="dropdownMenuLink"
 data-toggle="dropdown"
 aria-haspopup="true"
 aria-expanded="false"
 >
 {%{{ defaultLanguage$ | async }}%}

 <div
 class="dropdown-menu dropdown-menu-right border-0 shadow-sm"
 aria-labelledby="dropdownMenuLink"
 [class.d-block]="smallScreen && languageDropdown.isOpen()"
 >
 <a
 *ngFor="let lang of dropdownLanguages$ | async"
 href="javascript:void(0)"
 class="dropdown-item"
 (click)="onChangeLang(lang.cultureName)"
 >{{ lang?.displayName }}
 </div>
 </div>

 <li class="nav-item d-flex align-items-center">
 <ng-template #loginBtn>
 {{ AbpAccount::Login' | abpLocalization }}
 </ng-template>
 <div
 *ngIf="(currentUser$ | async)?.isAuthenticated; else loginBtn"
```

```

ngbDropdown
 class="dropdown"
 #currentUserDropdown="ngbDropdown"
 display="static"
>
<a
 ngbDropdownToggle
 class="nav-link"
 href="javascript:void(0)"
 role="button"
 id="dropdownMenuLink"
 data-toggle="dropdown"
 aria-haspopup="true"
 aria-expanded="false"
>
 <small *ngIf="(selectedTenant$ | async)?.name as tenantName"
 ><i>{{ tenantName }}</i>
 >\</small>
 >
 {{{ (currentUser$ | async)?.userName }}}

<div
 class="dropdown-menu dropdown-menu-right border-0 shadow-sm"
 aria-labelledby="dropdownMenuLink"
 [class.d-block]="smallScreen && currentUserDropdown.isOpen()"
>

 <i class="fa fa-cog mr-1"></i>{{{ 'AbpAccount::MyAccount' | abpLocalization }}}
 >

 <i class="fa fa-power-off mr-1"></i>{{{ 'AbpUi::Logout' | abpLocalization }}}
 >
</div>
</div>


```

```

Open `app.component.ts` in `src/app` folder and modify it as shown below:

```

```js
import { ..., ReplaceableComponentsService } from '@abp/ng.core'; // imported ReplaceableComponentsService
import { NavItemsComponent } from './nav-items/nav-items.component'; // imported NavItemsComponent
import { eThemeBasicComponents } from '@abp/ng.theme.basic'; // imported eThemeBasicComponents
//...
@Component({* component metadata */}

```

```

export class AppComponent implements OnInit {
 constructor(..., private replaceableComponents: ReplaceableComponentsService) {} // injected ReplaceableComponentsService

 ngOnInit() {
 //...

 this.replaceableComponents.add({
 component: NavItemsComponent,
 key: eThemeBasicComponents.NavItems,
 });
 }
}
```

```

The final UI looks like below:

![New nav-items](./images/replaced-nav-items-component.png)

See Also

- [How Replaceable Components Work with Extensions](./How-Replaceable-Components-Work-with-Extensions.md)
- [How to Replace PermissionManagementComponent](./Permission-Management-Component-Replacement.md)

10.3.5.6 Extensions

10.3.5.6.1 Overall

Angular UI Extensions

Angular UI extensions system allows you to add a new action to the actions menu, a new column to the data table, a new action to the toolbar of a page, and add a new field to the create and/or edit forms.

See the documents below for the details:

- * [Entity Action Extensions](Entity-Action-Extensions.md)
- * [Data Table Column (or Entity Prop) Extensions](Data-Table-Column-Extensions.md)
- * [Page Toolbar Extension](Page-Toolbar-Extensions.md)
- * [Dynamic Form (or Form Prop) Extensions](Dynamic-Form-Extensions.md)

Extensible Table Component

Using [ngx-datatable](<https://github.com/swimlane/ngx-datatable>) in extensible table.

```

```ts
<abp-extensible-table
 actionsText="Your Action"
 [data]="items"

```

```

[recordsTotal]="totalCount"
[actionsColumnWidth]="38"
[actionsTemplate]="customAction"
[list]="list"
(tableActivate)="onTableSelect($event)" >
</abp-extensible-table>
```
* `actionsText` : ** Column name of action column. **Type** : string
* `data` : Items shows in your table. **Type** : Array<any>
* `list` : Instance of ListService. **Type** : ListService
* `actionsColumnWidth` : Width of your action column. **Type** : number
* `actionsTemplate` : Template of the action when "click this button" or whatever.
Generally ng-template. **Type** : TemplateRef<any>
* `recordsTotal` : Count of the record total. **Type** : number
* `tableActivate` : The Output(). A cell or row was focused via the keyboard or a mouse click. **Type** : EventEmitter()

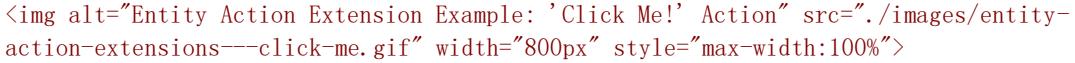
```

10.3.5.6.2 Entity Action Extensions

Entity Action Extensions for Angular UI

Introduction

Entity action extension system allows you to add a new action to the action menu for an entity. A "Click Me" action was added to the user management page below:



You can take any action (open a modal, make an HTTP API call, redirect to another page... etc) by writing your custom code. You can also access the current entity in your code.

How to Set Up

In this example, we will add a "Click Me!" action and alert the current row's `userName` in the user management page of the [Identity Module](../../Modules/Identity.md).

Step 1. Create Entity Action Contributors

The following code prepares a constant named `identityEntityActionContributors`, ready to be imported and used in your root module:

```

```js
// src/app/entity-action-contributors.ts

import {
 eIdentityComponents,
 IdentityEntityActionContributors,
 IdentityUserDto,
}

```

```

} from '@abp/ng.identity';
import { EntityAction, EntityActionList } from '@abp/ng.theme.shared/extensions';

const alertUserName = new EntityAction<IdentityUserDto>({
 text: 'Click Me!',
 action: data => {
 // Replace alert with your custom code
 alert(data.record.userName);
 },
 // See EntityActionOptions in API section for all options
});

export function alertUserNameContributor(actionList: EntityActionList<IdentityUserDto>) {
 actionList.addTail(alertUserName);
}

export const identityEntityActionContributors: IdentityEntityActionContributors = {
 // enum indicates the page to add contributors to
 [eIdentityComponents.Users]: [
 alertUserNameContributor,
 // You can add more contributors here
],
};

```

```

The list of actions, conveniently named as `actionList`, is a ****doubly linked list****. That is why we have used the `addTail` method, which adds the given value to the end of the list. You may find [\[all available methods here\]](#)(./Common/Utils/Linked-List.md).

Step 2. Import and Use Entity Action Contributors

Import `identityEntityActionContributors` in your routing module and pass it to the static `forLazy` method of `IdentityModule` as seen below:

```

```js
// src/app/app-routing.module.ts

// other imports
import { identityEntityActionContributors } from './entity-action-contributors';

const routes: Routes = [
 // other routes

 {
 path: 'identity',
 loadChildren: () =>
 import('@abp/ng.identity').then(m =>
 m.IdentityModule.forLazy({
 entityActionContributors: identityEntityActionContributors,
 })
),
 },
];

```

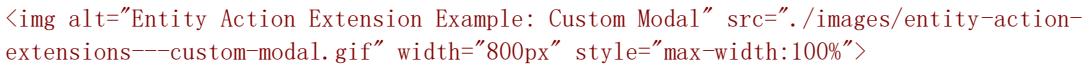
```
// other routes
];
```

```

That is it, `alertUserName` entity action will be added as the last action on the grid dropdown in the "Users" page (`UsersComponent`) of the `IdentityModule`.

How to Place a Custom Modal and Trigger It by Entity Actions

Let's employ dependency injection to extend the functionality of `IdentityModule` and add a quick view action for the User entity. We will take a lazy-loaded approach.

A screenshot of a browser window showing a modal dialog. The title bar says "Custom Modal". The content area contains the text "This is a custom modal triggered by an entity action." at the bottom right.

1. Create a folder at this path: `src/app/identity-extended`

2. Add an entity action similar to this:

```
```js
// src/app/identity-extended/entity-action-contributors.ts
```

```
import {
 eIdentityComponents,
 IdentityEntityActionContributors,
 IdentityUserDto,
} from '@abp/ng.identity';
import { EntityAction, EntityActionList } from '@abp/ng.theme.shared/extensions';
import { IdentityExtendedComponent } from './identity-extended.component';

const quickViewAction = new EntityAction<IdentityUserDto>({
 text: 'Quick View',
 action: data => {
 const component = data.getInjected(IdentityExtendedComponent);
 component.openUserQuickView(data.record);
 },
});

export function customModalContributor(actionList: EntityActionList<IdentityUserDto>) {
 actionList.addTail(quickViewAction);
}

export const identityEntityActionContributors: IdentityEntityActionContributors = {
 // enum indicates the page to add contributors to
 [eIdentityComponents.Users]: [
 customModalContributor,
 // You can add more contributors here
],
};
```

```

3. Create a parent component to the identity module.

```

```js
// src/app/identity-extended/identity-extended.component.ts

import { IdentityUserDto } from '@abp/ng.identity';
import { Component } from '@angular/core';

@Component({
 selector: 'app-identity-extended',
 templateUrl: './identity-extended.component.html',
})
export class IdentityExtendedComponent {
 isUserQuickViewVisible: boolean;

 user: IdentityUserDto;

 openUserQuickView(record: IdentityUserDto) {
 this.user = new Proxy(record, {
 get: (target, prop) => target[prop] || '-',
 });
 this.isUserQuickViewVisible = true;
 }
}
```

```

4. Add a router outlet and a modal to the parent component.

```

```html
<!-- src/app/identity-extended/identity-extended.component.html -->

<router-outlet></router-outlet>

<abp-modal [(visible)]="isUserQuickViewVisible">
 <ng-template #abpHeader>
 <h3>{{{{ user.userName }}}}</h3>
 </ng-template>

 <ng-template #abpBody>
 <table class="table table-borderless">
 <tbody>
 <tr>
 <th scope="row">{{{ 'AbpIdentity::DisplayName:Name' | abpLocalization }}}</th>
 <td>{{{{ user.name }}}}</td>
 </tr>
 <tr>
 <th scope="row">{{{ 'AbpIdentity::DisplayName:Surname' | abpLocalization }}}</th>
 <td>{{{{ user.surname }}}}</td>
 </tr>
 <tr>
 <th scope="row">{{{ 'AbpIdentity::EmailAddress' | abpLocalization }}}</th>
 <td>{{{{ user.email }}}}</td>
 </tr>
 </tbody>
 </table>
 </ng-template>
</abp-modal>

```

```

<tr>
 <th scope="row">{`{{ 'AbpIdentity::PhoneNumber' | abpLocalization }}`}</th>
 <td>{`{{ user.phoneNumber }}`}</td>
</tr>
</tbody>
</table>
</ng-template>

<ng-template #abpFooter>
 <button type="button" class="btn btn-secondary" abpClose>
 {`{{ 'AbpUi::Close' | abpLocalization }}`}
 </button>
</ng-template>
</abp-modal>
```

```

5. Add a module for the component and load `IdentityModule` as seen below:

```

```js
// src/app/identity-extended/identity-extended.module.ts

import { CoreModule } from '@abp/ng.core';
import { IdentityModule } from '@abp/ng.identity';
import { ThemeSharedModule } from '@abp/ng.theme.shared';
import { NgModule } from '@angular/core';
import { RouterModule } from '@angular/router';
import { identityEntityActionContributors } from './entity-action-contributors';
import { IdentityExtendedComponent } from './identity-extended.component';

@NgModule({
 imports: [
 CoreModule,
 ThemeSharedModule,
 RouterModule.forChild([
 {
 path: '',
 component: IdentityExtendedComponent,
 children: [
 {
 path: '',
 loadChildren: () =>
 IdentityModule.forLazy({
 entityActionContributors: identityEntityActionContributors,
 }),
 },
],
 },
]),
 declarations: [IdentityExtendedComponent],
 })
export class IdentityExtendedModule {}
```

```

6. Load `IdentityExtendedModule` instead of `IdentityModule` in your root routing module.

```
```js
// src/app/app-routing.module.ts

const routes: Routes = [
 // other routes

 {
 path: 'identity',
 loadChildren: () =>
 import('./identity-extended/identity-extended.module')
 .then(m => m.IdentityExtendedModule),
 },
 // other routes
];
````
```

That's it. As you see, we reached the `IdentityExtendedComponent` through dependency injection and called one of its methods in our action. The specific user was also available via `data.record`, so we were able to display a summary view.

API

ActionData<R = any>

`ActionData` is the shape of the parameter passed to all callbacks or predicates in an `EntityAction`.

It has the following properties:

- ****record**** is the row data, i.e. current value rendered in the table.

```
```js
{
 text: 'Click Me!',
 action: data => {
 alert(data.record.userName);
 },
}
````
```

- ****index**** is the table index where the record is at.

- ****getInjected**** is the equivalent of `[Injector.get](https://angular.io/api/core/Injector#get)`. You can use it to reach injected dependencies of `GridActionsComponent`, including, but not limited to, its parent component.

```
```js
{
````
```

```

    text: 'Click Me!',
    action: data => {
      const restService = data.getInjected(RestService);

      // Use restService public props and methods here
    },
    visible: data => {
      const usersComponent = data.getInjected/UsersComponent);

      // Use usersComponent public props and methods here
    },
}
```

```

### ### ActionCallback<T, R = any>

`ActionCallback` is the type of the callback function that can be passed to an `EntityAction` as `action` parameter. An action callback gets a single parameter, the `ActionData`. The return type may be anything, including `void`. Here is a simplified representation:

```

```js
type ActionCallback<T, R = any> = (data?: ActionData<T>) => R;
```

```

### ### ActionPredicate<T>

`ActionPredicate` is the type of the predicate function that can be passed to an `EntityAction` as `visible` parameter. An action predicate gets a single parameter, the `ActionData`. The return type must be `boolean`. Here is a simplified representation:

```

```js
type ActionPredicate<T> = (data?: ActionData<T>) => boolean;
```

```

### ### EntityActionOptions<R = any>

`EntityActionOptions` is the type that defines required and optional properties you have to pass in order to create an entity action.

Its type definition is as follows:

```

```js
type EntityActionOptions<R = any> = {
  action: ActionCallback<R>,
  text: string,
  icon?: string,
  permission?: string,
  visible?: ActionPredicate<R>,
  btnClass?: string,
  btnStyle?: string,
};
```

```

```
```
```

As you see, passing `action` and `text` is enough to create an entity action. Here is what each property is good for:

- ****action**** is a callback that is called when the grid action is clicked. (*_required_*)
- ****text**** is the button text which will be localized. (*_required_*)
- ****icon**** is the classes that define an icon to be placed before the text. (*_default:_ ````*)
- ****permission**** is the permission context which will be used to decide if this type of grid action should be displayed to the user or not. (*_default:_ ``undefined``*)
- ****visible**** is a predicate that will be used to decide if the current record should have this grid action or not. (*_default:_ ``() => true``*)
- ****btnClass**** is the classes that will be applied to the button. (*_default:_ ``'btn btn-primary text-center'``*)
- ****btnStyle**** is the styles that will be applied to the button. (*_default:_ ``````*)

You may find a full example below.

```
### EntityAction<R = any>
```

`EntityAction` is the class that defines your entity actions. It takes an `EntityActionOptions` and sets the default values to the properties, creating an entity action that can be passed to an entity contributor.

```
```js
const options: EntityActionOptions<IdentityUserDto> = {
 action: data => {
 const component = data.getInjected(IdentityExtendedComponent);
 component.unlock(data.record.id);
 },
 text: 'AbpIdentity::Unlock',
 icon: 'fa fa-unlock',
 permission: 'AbpIdentity.Users.Update',
 visible: data => data.record.isLockedOut,
 btnClass:'btn btn-warning text-center',
 btnStyle: '', //Adds inline style
};

const action = new EntityAction(options);
```
```

It also has two static methods to create its instances:

- ****EntityAction.create<R = any>(options: EntityActionOptions<R>)**** is used to create an instance of `EntityAction`.

```
```js
const action = EntityAction.create(options);
```
```

- ****EntityAction.createMany<R = any>(options: EntityActionOptions<R>[])**** is used to create multiple instances of `EntityAction` with given array of `EntityActionOptions`.

```
```js
```

```
const actions = EntityAction.createMany(optionsArray);
````
```

```
### EntityActionList<R = any>
```

`EntityActionList` is the list of actions passed to every action contributor callback as the first parameter named `actionList`. It is a ****doubly linked list****. You may find [all available methods here](./Common/Utils/Linked-List.md).

The items in the list will be displayed according to the linked list order, i.e. from head to tail. If you want to re-order them, all you have to do is something like this:

```
```js
export function reorderUserContributors(
 actionList: EntityActionList<IdentityUserDto>,
) {
 // drop "Unlock" button
 const unlockActionNode = actionList.dropByValue(
 'AbpIdentity:Unlock',
 (action, text) => action.text === text,
);

 // add it back to the head of the list
 actionList.addHead(unlockActionNode.value);
}
````
```

```
### EntityActionContributorCallback<R = any>
```

`EntityActionContributorCallback` is the type that you can pass as entity action contributor callbacks to static `forLazy` methods of the modules.

```
```js
// lockUserContributor should have EntityActionContributorCallback<IdentityUserDto> type

export function lockUserContributor(
 actionList: EntityActionList<IdentityUserDto>,
) {
 // add lockUser as 3rd action
 actionList.add(lockUser).byIndex(2);
}

export const identityEntityActionContributors = [
 [eIdentityComponents.Users]: [lockUserContributor],
];
````
```

See Also

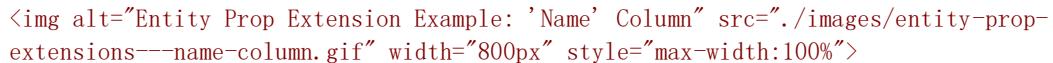
- [Customizing Application Modules Guide](./../Customizing-Application-Modules-Guide.md)

10.3.5.6.3 Data Table Column Extensions

```
# Data Table Column (or Entity Prop) Extensions for Angular UI
```

Introduction

Entity prop extension system allows you to add a new column to the data table for an entity or change/remove an already existing one. A "Name" column was added to the user management page below:



```

```

You will have access to the current entity in your code and display its value, make the column sortable, perform visibility checks, and more. You can also render custom HTML in table cells.

How to Set Up

In this example, we will add a "Name" column and display the value of the `name` field in the user management page of the [Identity Module](../../../../Modules/Identity.md).

Step 1. Create Entity Prop Contributors

The following code prepares a constant named `identityEntityPropContributors`, ready to be imported and used in your root module:

```
```js
// src/app/entity-prop-contributors.ts

import {
 eIdentityComponents,
 IdentityEntityPropContributors,
 IdentityUserDto,
} from '@abp/ng.identity';
import { EntityProp, EntityPropList, ePropType } from '@abp/ng.theme.shared/extensions';

const nameProp = new EntityProp<IdentityUserDto>({
 type: ePropType.String,
 name: 'name',
 displayName: 'AbpIdentity::Name',
 sortable: true,
 columnWidth: 250,
});

export function namePropContributor(propList: EntityPropList<IdentityUserDto>) {
 propList.addAfter(nameProp, 'userName', (value, name) => value.name === name);
}

export const identityEntityPropContributors: IdentityEntityPropContributors = {
```

```
// enum indicates the page to add contributors to
[eIdentityComponents.Users]: [
 namePropContributor,
 // You can add more contributors here
],
};

```

```

The list of props, conveniently named as `propList`, is a ****doubly linked list****. That is why we have used the `addAfter` method, which adds a node with given value after the first node that has the previous value. You may find [all available methods here](./Common/Utils/Linked-List.md).

Step 2. Import and Use Entity Prop Contributors

Import `identityEntityPropContributors` in your routing module and pass it to the static `forLazy` method of `IdentityModule` as seen below:

```
```js
// src/app/app-routing.module.ts

// other imports
import { identityEntityPropContributors } from './entity-prop-contributors';

const routes: Routes = [
 // other routes

 {
 path: 'identity',
 loadChildren: () =>
 import('@abp/ng.identity').then(m =>
 m.IdentityModule.forLazy({
 entityPropContributors: identityEntityPropContributors,
 })
),
 },
 // other routes
];
```

```

That is it, `nameProp` entity prop will be added, and you will see the "Name" column next to the usernames on the grid in the users page (`UsersComponent`) of the `IdentityModule`.

How to Render Custom HTML in Cells

You can use the `valueResolver` to render an HTML string in the table. Imagine we want to show a red times icon (X) next to unconfirmed emails and phones, instead of showing a green check icon next to confirmed emails and phones. The contributors below would do that for you.


```

```js
// src/app/entity-prop-contributors.ts

import {
 eIdentityComponents,
 IdentityEntityPropContributors,
 IdentityUserDto,
} from '@abp/ng.identity';
import { EntityProp, EntityPropList } from '@abp/ng.theme.shared/extensions';
import { of } from 'rxjs';

export function emailPropContributor(propList: EntityPropList<IdentityUserDto>) {
 const index = propList.indexOf('email', (value, name) => value.name === name);
 const droppedNode = propList.dropByIndex(index);
 const emailProp = new EntityProp<IdentityUserDto>({
 ...droppedNode.value,
 valueResolver: data => {
 const { email, emailConfirmed } = data.record;
 const icon = email && !emailConfirmed ? `<i class="fa fa-times text-danger ml-1"></i>` : '';
 return of((email || '') + icon); // should return an observable
 },
 });
 propList.addByIndex(emailProp, index);
}

export function phonePropContributor(propList: EntityPropList<IdentityUserDto>) {
 const index = propList.indexOf('phoneNumber', (value, name) => value.name === name);
 const droppedNode = propList.dropByIndex(index);
 const phoneProp = new EntityProp<IdentityUserDto>({
 ...droppedNode.value,
 valueResolver: data => {
 const { phoneNumber, phoneNumberConfirmed } = data.record;
 const icon =
 phoneNumber && !phoneNumberConfirmed ? `<i class="fa fa-times text-danger ml-1"></i>` : '';
 return of((phoneNumber || '') + icon); // should return an observable
 },
 });
 propList.addByIndex(phoneProp, index);
}

export const identityEntityPropContributors: IdentityEntityPropContributors = [
 [eIdentityComponents.Users]: [emailPropContributor, phonePropContributor],
];
```

```

```
```
```

> The `valueResolver` method should return an observable. You can wrap your return values with `of` from RxJS for that.

## ## Object Extensions

Extra properties defined on an existing entity will be included in the table based on their configuration. The values will also be mapped to and from `extraProperties` automatically. They are available when defining custom contributors, so you can drop, modify, or reorder them. The `isExtra` identifier will be set to `true` for these properties and will define this automatic behavior.

## ## API

### #### PropData<R = any>

`PropData` is the shape of the parameter passed to all callbacks or predicates in an `EntityProp`.

It has the following properties:

- **\*\*record\*\*** is the row data, i.e. current value rendered in the table.

```
```js
{
  type: ePropType.String,
  name: 'name',
  valueResolver: data => {
    const name = data.record.name || '';
    return of(name.toUpperCase());
  },
}
```

```

- **\*\*index\*\*** is the table index where the record is at.

- **\*\*getInjected\*\*** is the equivalent of [\[Injector.get\]](#) (<https://angular.io/api/core/Injector#get>). You can use it to reach injected dependencies of `ExtensibleTableComponent`, including, but not limited to, its parent component.

```
```js
{
  type: ePropType.String,
  name: 'name',
  valueResolver: data => {
    const restService = data.getInjected(RestService);
    const usersComponent = data.getInjected(UsersComponent);

    // Use restService and usersComponent public props and methods here
  }
}
```

```

```
 },
}
```

### ### PropCallback<T, R = any>

`PropCallback` is the type of the callback function that can be passed to an `EntityProp` as `prop` parameter. A prop callback gets a single parameter, the `PropData`. The return type may be anything, including `void`. Here is a simplified representation:

```
```js
type PropCallback<T, R = any> = (data?: PropData<T>) => R;
```

```

### ### PropPredicate<T>

`PropPredicate` is the type of the predicate function that can be passed to an `EntityProp` as `visible` parameter. A prop predicate gets a single parameter, the `PropData`. The return type must be `boolean`. Here is a simplified representation:

```
```js
type PropPredicate<T> = (data?: PropData<T>) => boolean;
```

```

### ### EntityPropOptions<R = any>

`EntityPropOptions` is the type that defines required and optional properties you have to pass in order to create an entity prop.

Its type definition is as follows:

```
```js
type EntityPropOptions<R = any> = {
  type: ePropType;
  name: string;
  displayName?: string;
  valueResolver?: PropCallback<R, Observable<any>>;
  sortable?: boolean;
  columnWidth?: number;
  permission?: string;
  visible?: PropPredicate<R>;
};
```

```

As you see, passing `type` and `name` is enough to create an entity prop. Here is what each property is good for:

- **\*\*type\*\*** is the type of the prop value. It is used for custom rendering in the table.  
(\_required\_)
- **\*\*name\*\*** is the property name (or key) which will be used to read the value of the prop.  
(\_required\_)

- **\*\*displayName\*\*** is the name of the property which will be localized and shown as column header. (`_default:_ `options.name``)
- **\*\*valueResolver\*\*** is a callback that is called when the cell is rendered. It must return an observable. (`_default:_ `data => of(data.record[options.name])``)
- **\*\*sortable\*\*** defines if the table is sortable based on this entity prop. Sort icons are shown based on it. (`_default:_ `false``)
- **\*\*columnWidth\*\*** defines a minimum width for the column. Good for horizontal scroll. (`_default:_ `undefined``)
- **\*\*permission\*\*** is the permission context which will be used to decide if a column for this entity prop should be displayed to the user or not. (`_default:_ `undefined``)
- **\*\*visible\*\*** is a predicate that will be used to decide if this entity prop should be displayed on the table or not. (`_default:_ `() => true``)

➤ Important Note: Do not use record in visibility predicates. First of all, the table header checks it too and the record will be ``undefined``. Second, if some cells are displayed and others are not, the table will be broken. Use the ``valueResolver`` and render an empty cell when you need to hide a specific cell.

You may find a full example below.

```
EntityProp<R = any>
```

``EntityProp`` is the class that defines your entity props. It takes an ``EntityPropOptions`` and sets the default values to the properties, creating an entity prop that can be passed to an entity contributor.

```
```js
const options: EntityPropOptions<IdentityUserDto> = {
  type: ePropType.String,
  name: 'email',
  displayName: 'AbpIdentity::EmailAddress',
  valueResolver: data => {
    const { email, emailConfirmed } = data.record;

    return of(
      (email || '') + (emailConfirmed ? `<i class="fa fa-check text-success ml-1"></i>` : ''),
    );
  },
  sortable: true,
  columnWidth: 250,
  permission: 'AbpIdentity.Users.ReadSensitiveData', // hypothetical
  visible: data => {
    const store = data.getInjected(Store);
    const selectSensitiveDataVisibility = ConfigState.getSetting(
      'Abp.Identity.IsSensitiveDataVisible' // hypothetical
    );

    return store.selectSnapshot(selectSensitiveDataVisibility).toLowerCase() === 'true';
  }
};
```

```
const prop = new EntityProp(options);  
```
```

It also has two static methods to create its instances:

- **\*\*EntityProp.create<R = any>(options: EntityPropOptions<R>)\*\*** is used to create an instance of `EntityProp`.

```
```js  
const prop = EntityProp.create(options);  
```
```

- **\*\*EntityProp.createMany<R = any>(options: EntityPropOptions<R>[])\*\*** is used to create multiple instances of `EntityProp` with given array of `EntityPropOptions`.

```
```js  
const props = EntityProp.createMany(optionsArray);  
```
```

```
EntityPropList<R = any>
```

`EntityPropList` is the list of props passed to every prop contributor callback as the first parameter named `propList`. It is a **\*doubly linked list\***. You may find [all available methods here](./Common/Utils/Linked-List.md).

The items in the list will be displayed according to the linked list order, i.e. from head to tail. If you want to re-order them, all you have to do is something like this:

```
```js  
export function reorderUserContributors(  
    propList: EntityPropList<IdentityUserDto>,  
) {  
    // drop email node  
    const emailPropNode = propList.dropByValue(  
        'AbpIdentity::EmailAddress',  
        (prop, text) => prop.text === text,  
    );  
  
    // add it back after phoneNumber  
    propList.addAfter(  
        emailPropNode.value,  
        'phoneNumber',  
        (value, name) => value.name === name,  
    );  
}
```

```
### EntityPropContributorCallback<R = any>
```

`EntityPropContributorCallback` is the type that you can pass as entity prop contributor callbacks to static `forLazy` methods of the modules.

```
```js  
export function isLockedOutPropContributor(
 propList: EntityPropList<IdentityUserDto>,
```

```

) {
 // add isLockedOutProp as 2nd column
 propList.add(isLockedOutProp).byIndex(1);
 }

export const identityEntityPropContributors = {
 [eIdentityComponents.Users]: [isLockedOutPropContributor],
};

```

```

See Also

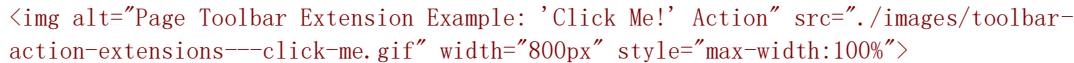
- [\[Customizing Application Modules Guide\]\(../../../../Customizing-Application-Modules-Guide.md\)](#)

10.3.5.6.4 Page Toolbar Extensions

Page Toolbar Extensions for Angular UI

Introduction

Page toolbar extension system allows you to add a new action to the toolbar of a page. A "Click Me" action was added to the user management page below:



The image shows a screenshot of a web application's user management page. At the top, there is a toolbar with several standard icons like search, refresh, and filter. Below the toolbar, there is a list of users. On the far right of this list, there is a blue button with the text "Click Me!". This button is the custom toolbar extension mentioned in the text.

You can take any action (open a modal, make an HTTP API call, redirect to another page... etc) by writing your custom code. You can also access to page data (the main record, usually an entity list) in your code. Additionally, you can pass in custom components instead of using the default button.

How to Add an Action to Page Toolbar

In this example, we will add a "Click Me!" action and log `userName` of all users in the user management page of the [\[Identity Module\]\(../../../../Modules/Identity.md\)](#) to the console.

Step 1. Create Toolbar Action Contributors

The following code prepares a constant named `identityToolbarActionContributors`, ready to be imported and used in your root module:

```

```js
// src/app/toolbar-action-contributors.ts

import {
 eIdentityComponents,
 IdentityToolbarActionContributors
} from '@abp/ng.identity';
import { IdentityUserDto } from '@abp/ng.identity/proxy';
import { ToolbarAction, ToolbarActionList } from '@abp/ng.theme.shared/extensions';

```

```

const logUserNames = new ToolbarAction<IdentityUserDto[]>({
 text: 'Click Me!',
 action: data => {
 // Replace log with your custom code
 data.record.forEach(user => console.log(user.userName));
 },
 // See ToolbarActionOptions in API section for all options
});

export function logUserNamesContributor(actionList: ToolbarActionList<IdentityUserDto[]>)
{
 actionList.addHead(logUserNames);
}

export const identityToolbarActionContributors: IdentityToolbarActionContributors = {
 // enum indicates the page to add contributors to
 [eIdentityComponents.Users]: [
 logUserNamesContributor,
 // You can add more contributors here
],
};

```

```

The list of actions, conveniently named as `actionList`, is a ****doubly linked list****. That is why we have used the `addHead` method, which adds the given value to the beginning of the list. You may find [\[all available methods here\]](#)(../Common/Utils/Linked-List.md).

Step 2. Import and Use Toolbar Action Contributors

Import `identityToolbarActionContributors` in your routing module and pass it to the static `forLazy` method of `IdentityModule` as seen below:

```

```js
// src/app/app-routing.module.ts

// other imports
import { identityToolbarActionContributors } from './toolbar-action-contributors';

const routes: Routes = [
 // other routes

 {
 path: 'identity',
 loadChildren: () =>
 import('@abp/ng.identity').then(m =>
 m.IdentityModule.forLazy({
 toolbarActionContributors: identityToolbarActionContributors,
 })
),
 },
];

```

```
// other routes
];
```

```

That is it, `logUserNames` toolbar action will be added as the first action on the page toolbar in the users page (`UsersComponent`) of the `IdentityModule`.

How to Add a Custom Component to Page Toolbar

In this example, we will add a custom "Click Me!" button and log `userName` of all users in the user management page of the [Identity Module](../../../../Modules/Identity.md) to the console.

A screenshot of a web browser showing a toolbar with a custom button labeled "Click Me!". The button is blue with white text and has a slight shadow. The background of the toolbar is light gray. To the right of the button, there is some placeholder text and a small icon.

Step 1. Create A Custom Component

We need to have a component before we can pass it to the toolbar action contributors:

```
```js
// src/app/click-me-button.component.ts

import { IdentityUserDto } from '@abp/ng.identity/proxy';
import { ActionData, EXTENSIONS_ACTION_DATA } from '@abp/ng.theme.shared/extensions';
import { Component, Inject } from '@angular/core';

@Component({
 selector: 'app-click-me-button',
 template: `<Click Me!>`,
})
export class ClickMeButtonComponent {
 constructor(
 @Inject(EXTENSIONS_ACTION_DATA)
 private data: ActionData<IdentityUserDto[]>
) {}

 handleClick() {
 this.data.record.forEach(user => console.log(user.userName));
 }
}
```

```

Here, `EXTENSIONS_ACTION_DATA` token provides us the context from the page toolbar. Therefore, we are able to reach the page data via `record`, which is an array of users, i. e. `IdentityUserDto[]`.

> We could also import `EXTENSIONS_ACTION_CALLBACK` from **@abp/ng.theme.shared/extensions** package, which is a higher order function that triggers the predefined `action` when called. It passes `ActionData` as the first

parameter, so you do not have to pass it explicitly. In other words, `EXTENSIONS_ACTION_CALLBACK` can be called without any parameters and it will not fail.

Step 2. Create Toolbar Action Contributors

The following code prepares a constant named `identityToolbarActionContributors`, ready to be imported and used in your root module. When `ToolbarComponent` is used instead of `ToolbarAction`, we can pass a component in:

```
```js
// src/app/toolbar-action-contributors.ts

import {
 eIdentityComponents,
 IdentityToolbarActionContributors
} from '@abp/ng.identity';
import { IdentityUserDto } from '@abp/ng.identity/proxy';
import { ToolbarActionList, ToolbarComponent } from '@abp/ng.theme.shared/extensions';
import { ClickMeButtonComponent } from './click-me-button.component';

const logUserNames = new ToolbarComponent<IdentityUserDto[]>({
 component: ClickMeButtonComponent,
 // See ToolbarActionOptions in API section for all options
});

export function logUserNamesContributor(actionList: ToolbarActionList<IdentityUserDto[]>)
{
 actionList.addHead(logUserNames);
}

export const identityToolbarActionContributors: IdentityToolbarActionContributors = {
 // enum indicates the page to add contributors to
 [eIdentityComponents.Users]: [
 logUserNamesContributor,
 // You can add more contributors here
],
};

```

```

The list of actions, conveniently named as `actionList`, is a ****doubly linked list****. That is why we have used the `addHead` method, which adds the given value to the beginning of the list. You may find [\[all available methods here\]](#)(..../Common/Utils/Linked-List.md).

Step 3. Import and Use Toolbar Action Contributors

Import `identityToolbarActionContributors` in your routing module and pass it to the static `forLazy` method of `IdentityModule` as seen below.

```
```js
// src/app/app-routing.module.ts
```

```

// other imports
import { identityToolbarActionContributors } from './toolbar-action-contributors';

const routes: Routes = [
 // other routes

 {
 path: 'identity',
 loadChildren: () =>
 import('@abp/ng.identity').then(m =>
 m.IdentityModule.forChild({
 toolbarActionContributors: identityToolbarActionContributors,
 })
),
 },
 // other routes
];
```

```

That is it, `logUserNames` toolbar action will be added as the first action on the page toolbar in the users page (`UsersComponent`) of the `IdentityModule` and it will be triggered by a custom button, i.e. `ClickMeButtonComponent`. Please note that ****component projection is not limited to buttons**** and you may use other UI components.

How to Place a Custom Modal and Trigger It by Toolbar Actions

Please check the same topic in [[entity action extensions document](#)] (Entity-Action-Extensions.md) and replace entity action with a toolbar action.

API

ActionData<R = any>

`ActionData` is the shape of the parameter passed to all callbacks or predicates in a `ToolbarAction` .

It has the following properties:

- ****record**** is the page data, the main record on a page, usually an entity list (e.g. list of users).

```

```js
{
 text: 'Click Me!',
 action: data => {
 data.record.forEach(user => {
 console.log(user.userName);
 });
 }
}
```

```

- ****getInjected**** is the equivalent of `[Injector.get](https://angular.io/api/core/Injector#get)`. You can use it to reach injected dependencies of `'PageToolbarComponent'`, including, but not limited to, its parent component.

```
```js
{
 text: 'Click Me!',
 action: data => {
 const restService = data.getInjected(RestService);

 // Use restService public props and methods here
 },
 visible: data => {
 const usersComponent = data.getInjected/UsersComponent);

 // Use usersComponent public props and methods here
 },
}
````
```

ActionCallback<T, R = any>

`'ActionCallback'` is the type of the callback function that can be passed to a `'ToolbarAction'` as `'action'` parameter. An action callback gets a single parameter, the `'ActionData'`. The return type may be anything, including `'void'`. Here is a simplified representation:

```
```js
type ActionCallback<T, R = any> = (data?: ActionData<T>) => R;
````
```

ActionPredicate<T>

`'ActionPredicate'` is the type of the predicate function that can be passed to a `'ToolbarAction'` as `'visible'` parameter. An action predicate gets a single parameter, the `'ActionData'`. The return type must be `'boolean'`. Here is a simplified representation:

```
```js
type ActionPredicate<T> = (data?: ActionData<T>) => boolean;
````
```

ToolbarActionOptions<R = any>

`'ToolbarActionOptions'` is the type that defines required and optional properties you have to pass in order to create an toolbar action.

Its type definition is as follows:

```
```js
type ToolbarActionOptions<R = any> = {
```

```

 action: ActionCallback<R>,
 text: string,
 icon?: string,
 permission?: string,
 visible?: ActionPredicate<R>,
 };
```

```

As you see, passing `action` and `text` is enough to create an toolbar action. Here is what each property is good for:

- ****action**** is a callback that is called when the toolbar action is clicked. (*_required_*)
- ****text**** is the button text which will be localized. (*_required_*)
- ****icon**** is the classes that define an icon to be placed before the text. (*_default:_`''`*)
- ****permission**** is the permission context which will be used to decide if this toolbar action should be displayed to the user or not. (*_default:_`undefined`*)
- ****visible**** is a predicate that will be used to decide if the page toolbar should have this action or not. (*_default:_`() => true`*)

You may find a full example below.

```
### ToolbarAction<R = any>
```

`ToolbarAction` is the class that defines your toolbar actions. It takes an `ToolbarActionOptions` and sets the default values to the properties, creating an toolbar action that can be passed to an toolbar contributor.

```

```js
const options: ToolbarActionOptions<IdentityUserDto[]> = {
 action: data => {
 const service = data.getInjected(MyCustomIdentityService);
 const lockedUsers = data.record.filter(user => user.isLockedOut);
 service.unlockAll(lockedUsers);
 },
 text: 'MyProjectName::UnlockAll',
 icon: 'fa fa-unlock',
 permission: 'AbpIdentity.Users.Update',
 visible: data => data.record.some(user => user.isLockedOut),
};

const action = new ToolbarAction(options);
```

```

It also has two static methods to create its instances:

- ****ToolbarAction.create<R = any>(options: ToolbarActionOptions<R>)**** is used to create an instance of `ToolbarAction`.

```

```js
const action = ToolbarAction.create(options);
```

```

- **ToolbarAction.createMany**`<R = any>\(options: ToolbarActionOptions<R>\[]\)` is used to create multiple instances of `ToolbarAction` with given array of `ToolbarActionOptions`.

```
### ToolbarComponentOptions<R = any>
```

`ToolbarComponentOptions` is the type that defines required and optional properties you have to pass in order to create an toolbar component.

Its type definition is as follows:

```
```js
type ToolbarComponentOptions<R = any> = {
 component: Type<any>,
 action?: ActionCallback<R>,
 permission?: string,
 visible?: ActionPredicate<R>,
};```

```

As you see, passing `action` and `text` is enough to create an toolbar action. Here is what each property is good for:

- **component** is the constructor of the component to be projected. (*\_required\_*)
- **action** is a predefined callback that you can reach in your component via `EXTENSIONS_ACTION_CALLBACK` token and trigger. (*\_optional\_*)
- **permission** is the permission context which will be used to decide if this toolbar action should be displayed to the user or not. (*\_default:\_ undefined*)
- **visible** is a predicate that will be used to decide if the page toolbar should have this action or not. (*\_default:\_ () => true*)

You may find a full example below.

```
ToolbarComponent<R = any>
```

`ToolbarComponent` is the class that defines toolbar actions which project a custom component. It takes an `ToolbarComponentOptions` and sets the default values to the properties, creating a toolbar action that can be passed to an toolbar contributor.

```
```js
const options: ToolbarComponentOptions<IdentityUserDto[]> = {
  component: UnlockAllButton,
  action: data => {
    const service = data.getInjected(MyCustomIdentityService);
    const lockedUsers = data.record.filter(user => user.isLockedOut);
    service.unlockAll(lockedUsers);
  },
  permission: 'AbpIdentity.Users.Update',
  visible: data => data.record.some(user => user.isLockedOut),
};

const action = new ToolbarComponent(options);```

```

```
```
```

It also has two static methods to create its instances:

- **ToolbarComponent.create<R = any>(options: ToolbarComponentOptions<R>)** is used to create an instance of `ToolbarComponent`.

```
```js
const action = ToolbarComponent.create(options);
````
```

- **ToolbarComponent.createMany<R = any>(options: ToolbarComponentOptions<R>[])** is used to create multiple instances of `ToolbarComponent` with given array of `ToolbarComponentOptions`.

```
```js
const actions = ToolbarComponent.createMany(optionsArray);
````
```

#### ### ToolbarActionList<R = any>

`ToolbarActionList` is the list of actions passed to every action contributor callback as the first parameter named `actionList`. It is a **doubly linked list**. You may find [all available methods here](../Common/Utils/Linked-List.md).

The items in the list will be displayed according to the linked list order, i.e. from head to tail. If you want to re-order them, all you have to do is something like this:

```
```js
export function reorderUserContributors(
  actionList: ToolbarActionList<IdentityUserDto[]>,
) {
  // drop "New User" button
  const newUserActionNode = actionList.dropByValue(
    'AbpIdentity:::NewUser',
    (action, text) => action['text'] === text,
  );

  // add it back to the head of the list
  actionList.addHead(newUserActionNode.value);
}

export const identityEntityActionContributors = [
  [eIdentityComponents.Users]: [
    logUserNamesContributor,
    reorderUserContributors,
  ],
];
````
```

#### ### ToolbarActionContributorCallback<R = any>

`ToolbarActionContributorCallback` is the type that you can pass as toolbar action contributor callbacks to static `forLazy` methods of the modules.

```

```js
// exportUsersContributor should have ToolbarActionContributorCallback<IdentityUserDto[]>
type

export function exportUsersContributor(
  actionList: ToolbarActionList<IdentityUserDto[]>,
) {
  // add exportUsers just before the last action
  actionList.add(exportUsers).byIndex(-1);
}

export const identityEntityActionContributors = {
  [eIdentityComponents.Users]: [exportUsersContributor],
};

```

```

## ## See Also

- [\[Customizing Application Modules Guide\]\(../../../../Customizing-Application-Modules-Guide.md\)](#)

### 10.3.5.6.5 Dynamic Form Extensions

#### # Dynamic Form (or Form Prop) Extensions for Angular UI

##### ## Introduction

Form prop extension system allows you to add a new field to the create and/or edit forms for a form or change/remove an already existing one. A "Date of Birth" field was added to the user management page below:



The image shows a screenshot of a user management application. It features a table with columns for 'Name', 'Email', 'Role', and 'Actions'. In the 'Actions' column for a specific user, there is a 'Edit' button. When clicked, a modal dialog appears. Inside the modal, there are fields for 'Name', 'Email', and 'Role', along with a new 'Date of Birth' field, which is highlighted in red.

You can validate the field, perform visibility checks, and do more. You will also have access to the current entity when creating a contributor for an edit form.

##### ## How to Set Up

In this example, we will add a "Date of Birth" field in the user management page of the [\[Identity Module\]\(../../../../Modules/Identity.md\)](#) and validate it.

#### ### Step 1. Create Form Prop Contributors

The following code prepares two constants named `identityCreateFormPropContributors` and `identityEditFormPropContributors`, ready to be imported and used in your root module:

```

```js
// src/app/form-prop-contributors.ts

```

```

import {
  eIdentityComponents,
  IdentityCreateFormPropContributors,
} from '@abp/ng.identity';
import { IdentityUserDto } from '@abp/ng.identity/proxy';
import { ePropType, FormProp, FormPropList } from '@abp/ng.theme.shared/extensions';
import { Validators } from '@angular/forms';

const birthdayProp = new FormProp<IdentityUserDto>({
  type: ePropType.Date,
  name: 'birthday',
  displayName: 'AbpIdentity::Birthday',
  validators: () => [Validators.required],
});

export function birthdayPropContributor(propList: FormPropList<IdentityUserDto>) {
  propList.addByIndex(birthdayProp, 4);
}

export const identityCreateFormPropContributors: IdentityCreateFormPropContributors = {
  // enum indicates the page to add contributors to
  [eIdentityComponents.Users]: [
    birthdayPropContributor,
    // You can add more contributors here
  ],
};

export const identityEditFormPropContributors = identityCreateFormPropContributors;
// you may define different contributors for edit form if you like

```

```

The list of props, conveniently named as `propList`, is a **\*\*doubly linked list\*\***. That is why we have used the `addByIndex` method, which adds the given value to the specified index of the list. You may find [\[all available methods here\]](#)(..../Common/Utils/Linked-List.md).

#### ### Step 2. Import and Use Form Prop Contributors

Import `identityCreateFormPropContributors` and `identityEditFormPropContributors` in your routing module and pass it to the static `forLazy` method of `IdentityModule` as seen below:

```

```js
// src/app/app-routing.module.ts

// other imports
import {
  identityCreateFormPropContributors,
  identityEditFormPropContributors,
} from './form-prop-contributors';

```

```

const routes: Routes = [
  // other routes

  {
    path: 'identity',
    loadChildren: () =>
      import('@abp/ng.identity').then(m =>
        m.IdentityModule.forRoot({
          createFormPropContributors: identityCreateFormPropContributors,
          editFormPropContributors: identityEditFormPropContributors,
        })
      ),
  },
];

// other routes
];
```

```

That is it, `birthdayProp` form prop will be added, and you will see the datepicker for the "Date of Birth" field right before the "Email address" in the forms of the users page in the `IdentityModule`.

## ## Object Extensions

Extra properties defined on an existing entity will be included in the create and edit forms and validated based on their configuration. The form values will also be mapped to and from `extraProperties` automatically. They are available when defining custom contributors, so you can drop, modify, or reorder them. The `isExtra` identifier will be set to `true` for these properties and will define this automatic behavior.

## ## API

### ### PropData<R = any>

`PropData` is the shape of the parameter passed to all callbacks or predicates in a `FormProp`.

It has the following properties:

- **\*\*getInjected\*\*** is the equivalent of `[Injector.get](https://angular.io/api/core/Injector#get)`. You can use it to reach injected dependencies of `ExtensibleFormPropComponent`, including, but not limited to, its parent components.

```

```js
{
  type: ePropType.Enum,
  name: 'myField',
  options: data => {
    const restService = data.getInjected(RestService);
    const usersComponent = data.getInjected(UsersComponent);
  }
}
```

```

```
// Use restService and usersComponent public props and methods here
},
__,
}
```

- **\*\*record\*\*** is the row data, i.e. current value of the selected item to edit. This property is available only on edit forms.

```
```js
{
  type: ePropType.String,
  name: 'myProp',
  readonly: data => data.record.someOtherProp,
}
```

```

### PropCallback<T, R = any>

`PropCallback` is the type of the callback function that can be passed to a `FormProp` as `prop` parameter. A prop callback gets a single parameter, the `PropData`. The return type may be anything, including `void`. Here is a simplified representation:

```
```js
type PropCallback<T, R = any> = (data?: PropData<T>) => R;
```

```

### PropPredicate<T>

`PropPredicate` is the type of the predicate function that can be passed to a `FormProp` as `visible` parameter. A prop predicate gets a single parameter, the `PropData`. The return type must be `boolean`. Here is a simplified representation:

```
```js
type PropPredicate<T> = (data?: PropData<T>) => boolean;
```

```

### FormPropOptions<R = any>

`FormPropOptions` is the type that defines required and optional properties you have to pass in order to create a form prop.

Its type definition is as follows:

```
```js
type FormPropOptions<R = any> = {
  type: ePropType;
  name: string;
  displayName?: string;
  id?: string;
  permission?: string;
  visible?: PropPredicate<R>;
}
```

```

```

readonly?: PropPredicate<R>;
disabled?: PropPredicate<R>;
validators?: PropCallback<R, ValidatorFn[]>;
asyncValidators?: PropCallback<R, AsyncValidatorFn[]>;
defaultValue?: boolean | number | string | Date;
options?: PropCallback<R, Observable<ABP.Option<any>[]>>;
autocomplete?: string;
isExtra?: boolean;
};

```

```

As you see, passing `type` and `name` is enough to create a form prop. Here is what each property is good for:

- ****type**** is the type of the prop value. It defines which input is rendered for the prop in the form. (*_required_*)
- ****name**** is the property name (or key) which will be used to read the value of the prop. (*_required_*)
- ****displayName**** is the name of the property which will be localized and shown as column header. (*_default:_ `options.name`*)
- ****id**** will be set as the `for` attribute of the label and the `id` attribute of the input for the field. (*_default:_ `options.name`*)
- ****permission**** is the permission context which will be used to decide if a column for this form prop should be displayed to the user or not. (*_default:_ `undefined`*)
- ****visible**** is a predicate that will be used to decide if this prop should be displayed on the form or not. (*_default:_ `() => true`*)
- ****readonly**** is a predicate that will be used to decide if this prop should be readonly or not. (*_default:_ `() => false`*)
- ****disabled**** is a predicate that will be used to decide if this prop should be disabled or not. (*_default:_ `() => false`*)
- ****validators**** is a callback that returns validators for the prop. (*_default:_ `() => []`*)
- ****asyncValidators**** is a callback that returns async validators for the prop. (*_default:_ `() => []`*)
- ****defaultValue**** is the initial value the field will have. (*_default:_ `null`*)
- ****options**** is a callback that is called when a dropdown is needed. It must return an observable. (*_default:_ `undefined`*)
- ****autocomplete**** will be set as the `autocomplete` attribute of the input for the field. Please check [possible values] (<https://developer.mozilla.org/en-US/docs/Web/HTML/Attributes/autocomplete#Values>). (*_default:_ `off`*)
- ****isExtra**** indicates this prop is an object extension. When `true`, the value of the field will be mapped from and to `extraProperties` of the entity. (*_default:_ `undefined`*)

> Important Note: Do not use `record` property of `PropData` in create form predicates and callbacks, because it will be `undefined`. You can use it on edit form contributors though.

You may find a full example below.

```
### FormProp\<R = any\>
```

`FormProp` is the class that defines your form props. It takes a `FormPropOptions` and sets the default values to the properties, creating a form prop that can be passed to a form contributor.

```
```js
const options: FormPropOptions<IdentityUserDto> = {
 type: ePropType.Enum,
 name: 'myProp',
 displayName: 'Default::MyPropName',
 id: 'my-prop',
 permission: 'AbpIdentity.Users.ReadSensitiveData', // hypothetical
 visible: data => {
 const store = data.getInjected(Store);
 const selectSensitiveDataVisibility = ConfigState.getSetting(
 'Abp.Identity.IsSensitiveDataVisible' // hypothetical
);

 return store.selectSnapshot(selectSensitiveDataVisibility).toLowerCase() === 'true';
 },
 readonly: data => data.record.someProp,
 disabled: data => data.record.someOtherProp,
 validators: () => [Validators.required],
 asyncValidators: data => {
 const http = data.getInjected(HttpClient);

 function validate(control: AbstractControl): Observable<ValidationErrors | null> {
 if (control.pristine) return of(null);

 return http
 .get('https://api.my-brand.io/hypothetical/endpoint/' + control.value)
 .pipe(map(response => (response.valid ? null : { invalid: true })));
 }

 return [validate];
 },
 defaultValue: 0,
 options: data => {
 const service = data.getInjected(MyIdentityService);

 return service.getMyPropOptions()
 .pipe(
 map(({items}) => items.map(
 item => ({key: item.name, value: item.id })
)),
);
 },
 autocomplete: 'off',
 isExtra: true,
 template: undefined | Type<any> // Custom angular component
};

const prop = new FormProp(options);
```

```
```
```

FormProp has the template option since version 6.0. it can accept custom angular component.

The component can access PropData and Prop.

Example of the custom prop component.

```
```js
```

```
import {
 EXTENSIBLE_FORM_VIEW_PROVIDER,
 EXTENSIONS_FORM_PROP,
 EXTENSIONS_FORM_PROP_DATA,
} from '@abp/ng.theme.shared/extensions';
```

```
@Component({
 selector: 'my-custom-custom-prop',
 templateUrl: './my-custom-custom-prop.component.html',
 viewProviders: [EXTENSIBLE_FORM_VIEW_PROVIDER], //you should add this, otherwise form-group doesn't work.
})
export class MyCustomPropComponent {
 constructor(
 @Inject(EXTENSIONS_FORM_PROP) private formProp: FormProp,
 @Inject(EXTENSIONS_FORM_PROP_DATA) private propData: ProfileDto,
 ...)
 ...
}
```

It also has two static methods to create its instances:

- **\*\*FormProp.create<R = any>(options: FormPropOptions<R>)\*\*** is used to create an instance of `FormProp`.

```
```js
```

```
const prop = FormProp.create(options);  
```
```

- **\*\*FormProp.createMany<R = any>(options: FormPropOptions<R>[])\*\*** is used to create multiple instances of `FormProp` with given array of `FormPropOptions`.

```
```js
```

```
const props = FormProp.createMany(optionsArray);  
```
```

```
FormPropList<R = any>
```

`FormPropList` is the list of props passed to every prop contributor callback as the first parameter named `propList`. It is a **doubly linked list**. You may find [\[all available methods here\]](#) ([./Common/Utils/Linked-List.md](#)).

The items in the list will be displayed according to the linked list order, i.e. from head to tail. If you want to re-order them, all you have to do is something like this:

```
```js
```

```
export function reorderUserContributors()
```

```

    propList: FormPropList<IdentityUserDto>,
) {
    // drop email node
    const emailPropNode = propList.dropByValue(
        'AbpIdentity::EmailAddress',
        (prop, displayName) => prop.displayName === displayName,
    );
}

// add it back after phoneNumber
propList.addAfter(
    emailPropNode.value,
    'phoneNumber',
    (value, name) => value.name === name,
);
}
```

```

#### ### CreateFormPropContributorCallback<R = any>

`CreateFormPropContributorCallback` is the type that you can pass as **\*\*create form\*\*** prop contributor callbacks to static `forLazy` methods of the modules.

```

```js
export function myPropCreateContributor(
    propList: FormPropList<IdentityUserDto>,
) {
    // add myProp as 2nd field from the start
    propList.add(myProp).byIndex(1);
}

export const identityCreateFormPropContributors = {
    [eIdentityComponents.Users]: [myPropCreateContributor],
};
```

```

#### ### EditFormPropContributorCallback<R = any>

`EditFormPropContributorCallback` is the type that you can pass as **\*\*edit form\*\*** prop contributor callbacks to static `forLazy` methods of the modules.

```

```js
export function myPropEditContributor(
    propList: FormPropList<IdentityUserDto>,
) {
    // add myProp as 2nd field from the end
    propList.add(myProp).byIndex(-1);
}

export const identityEditFormPropContributors = {
    [eIdentityComponents.Users]: [myPropEditContributor],
};
```

```

## ## See Also

- [Customizing Application Modules Guide](../../../../Customizing-Application-Modules-Guide.md)

### 10.3.5.6.6 Date, time and datetime format pipes

```
{%{
DateTime Format Pipes
```

You can format date by Date pipe of angular.

#### Example

```
```html
<span>{{today | date 'dd/mm/yy'}}</span>
````
```

ShortDate, ShortTime and ShortDateTime format data like angular's date pipe but easier.  
Also the pipes get format from config service by culture.

#### # ShortDate Pipe

```
```html
<span> {{today | shortDate }}</span>
````
```

#### # ShortTime Pipe

```
```html
<span> {{today | shortTime }}</span>
````
```

#### # ShortDateTime Pipe

```
```html
<span> {{today | shortDateTime }}</span>
````
```

```
} %}
```

### 10.3.6 Components

#### 10.3.6.1 Page

##### # Page Component

ABP provides a component that wraps your content with some built-in components to reduce the amount of code you need to write.

If the template of a component looks as follows, you can utilize the `abp-page` component.

Let's look at the following example without `abp-page` component.

```
`dashboard.component.ts`

```html  
<div class="row entry-row">  
  <div class="col-auto">  
    <h1 class="content-header-title">{{{{ ::Dashboard' | abpLocalization }}}}</h1>  
  </div>  
  <div id="breadcrumb" class="col-lg-auto pl-lg-0">  
    <abp-breadcrumb></abp-breadcrumb>  
  </div>  
  <div class="col">  
    <abp-page-toolbar [record]="data"></abp-page-toolbar>  
  </div>  
</div>  
  
<div id="dashboard-id">  
  <!-- dashboard content here -->  
</div>  
```
```

## ## Page Parts

PageComponent divides the template shown above into three parts, `title`, `breadcrumb`, `toolbar`. Each can be configured separately. There, also, is an enum exported from the package that describes each part.

```
```javascript  
export enum PageParts {  
  title = 'PageTitleContainerComponent',  
  breadcrumb = 'PageBreadcrumbContainerComponent',  
  toolbar = 'PageToolbarContainerComponent',  
}  
  
// You can import this enum from -> import { PageParts } from '@abp/ng. components/page';  
```
```

## ## Usage

Firstly, you need to import `PageModule` from `@abp/ng. components/page` as follows:

```
`dashboard.module.ts`

```javascript  
import { PageModule } from '@abp/ng. components/page';  
import { DashboardComponent } from './dashboard.component';
```

```

@NgModule({
  declarations: [DashboardComponent],
  imports: [PageModule]
})
export class DashboardModule {}
```

```

And change the template of `dashboard.component.ts` to the following:

```

```html
<abp-page [title]="" ::Dashboard' | abpLocalization" [toolbar]="data">
  <div id="dashboard-id">
    <!-- .... -->
  </div>
</abp-page>
```

```

#### ## Inputs

- \* title: `string`: Will be be rendered within `h1.content-header-title`. If not provided, the parent `div` will not be rendered
- \* breadcrumb: `boolean`: Determines whether to render `abp-breadcrumb`. Default is `true`.
- \* toolbar: `any`: Will be passed into `abp-page-toolbar` component through `record` input. If your page does not contain `abp-page-toolbar`, you can simply omit this field.

#### ## Overriding template

If you need to replace the template of any part, you can use the following sub-components.

```

```html
<abp-page>
  <abp-page-title-container class="col">
    <h2>Custom Title</h2>
  </abp-page-title-container>

  <abp-page-breadcrumb-container class="col">
    <my-breadcrumb></my-breadcrumb>
  </abp-page-breadcrumb-container>

  <abp-page-toolbar-container class="col">
    <button (click)="doSth()">Some Action</button>
  </abp-page-toolbar-container>
</abp-page>
```

```

You do not have to provide them all. You can just use which one you need to replace. These components have priority over the inputs declared above. If you use these components, you can omit the inputs.

#### ## PagePartDirective

`PageModule` provides a structural directive that is used internally within `PageComponent` and can also be used externally.

`PageComponent` employs this directive internally as follows:

```
```html
<div class="col-lg-auto pl-lg-0" *abpPagePart="pageParts.breadcrumb">
  <abp-breadcrumb></abp-breadcrumb>
</div>
```

```

It also can take a context input as follows:

```
```html
<div class="col" *abpPagePart="pageParts.toolbar; context: toolbarData">
  <abp-page-toolbar [record]="toolbarData"></abp-page-toolbar>
</div>
```

```

Its render strategy can be provided through Angular's Dependency Injection system.

It expects a service through the `PAGE\_RENDER\_STRATEGY` injection token that implements the following interface.

```
```javascript
interface PageRenderStrategy {
  shouldRender(type?: string): boolean | Observable<boolean>;
  onInit?(type?: string, injector?: Injector, context?: any): void;
  onDestroy?(type?: string, injector?: Injector, context?: any): void;
  onContextUpdate?(change?: SimpleChange): void;
}
```

```

- \* `shouldRender` (required): It takes a string input named `type` and expects a `boolean` or `Observable<boolean>` in return.
- \* `onInit` (optional): Will be called when the directive is initiated. Three inputs will be passed into this method.
  - \* `type`: type of the page part
  - \* `injector`: injector of the directive which could be used to retrieve anything from directive's DI tree.
  - \* `context`: whatever context is available at the initialization phase.
- \* `onDestroy` (optional): Will be called when the directive is destroyed. The parameters are the same with `onInit`.
- \* `onContextUpdate` (optional): Will be called when the context is updated.
  - \* `change`: changes of the `context` will be passed through this method.

Let's see everything in action.

```
```javascript
import {
  PageModule,
  PageRenderStrategy,
```

```

    PageParts,
    PAGE_RENDER_STRATEGY
} from '@abp/ng. components/page';

@Injectable()
export class MyPageRenderStrategy implements PageRenderStrategy {
  shouldRender(type: string) {
    // meaning everything but breadcrumb and custom-part will be rendered
    return type !== PageParts.breadcrumb && type !== 'custom-part';
  }

  /**
   * shouldRender can also return an Observable<boolean> which means
   * an async service can be used within.
  */

  constructor(private service: SomeAsyncService) {}

  shouldRender(type: string) {
    return this.service.checkTypeAsync(type).pipe(map(val => val.isTrue()));
  }
}

onInit(type: string, injector: Injector, context: any) {
  // this method will be called in ngOnInit of the directive
}

onDestroy(type: string, injector: Injector, context: any) {
  // this method will be called in ngOnDestroy of the directive
}

onContextUpdate?(change?: SimpleChange) {
  // this method will be called everytime context is updated within the directive
}

@Component({
  selector: 'app-dashboard',
  template: `
    <abp-page [title]="" ::Dashboard' | abpLocalization">
      <abp-page-toolbar-container>
        <button>New Dashboard</button>
      </abp-page-toolbar-container>

      <div class="dashboard-content">
        <h3 *abpPagePart="custom-part"> Inner Title </h3>
      </div>
    </abp-page>
  `
})
export class DashboardComponent {}

@NgModule({

```

```

imports: [PageModule],
declarations: [DashboardComponent],
providers: [
  {
    provide: PAGE_RENDER_STRATEGY,
    useClass: MyPageRenderStrategy,
  }
]
})
export class DashboardModule {}
```

```

#### ## See Also

- [Page Toolbar Extensions for Angular UI](./Page-Page-Toolbar-Extensions.md)

#### 10.3.6.2 Chart

##### # Chart Component

ABP Chart component exposed by `@abp/ng.components/chart.js` is based on [`charts.js`](https://www.chartjs.org/) v3+. You don't need to install the `chart.js` package. Since the `@abp/ng.components` is dependent on the `chart.js`, the package is already installed in your project.

> Chart component loads `chart.js` script lazy. So it does not increase the bundle size.

##### ## How to Use

First of all, need to import the `ChartModule` to your feature module as follows:

```

```ts
// your-feature.module.ts

import { ChartModule } from '@abp/ng.components/chart.js';
import { ChartDemoComponent } from './chart-demo.component';

@NgModule({
  imports: [
    ChartModule,
    // ...
  ],
  declarations: [ChartDemoComponent],
  // ...
})
export class YourFeatureModule {}
```

```

Then, `abp-chart` component can be used. See an example:

```

```ts
// chart-demo.component.ts

import { Component } from '@angular/core';

@Component({
  selector: 'app-chart-demo',
  template: `<abp-chart type="pie" [data]="data"></abp-chart>`,
})
export class ChartDemoComponent {
  data = {
    labels: ['Data 1', 'Data 2', 'Data 3'],
    datasets: [
      {
        label: 'Dataset 1',
        data: [40, 15, 45],
        backgroundColor: ['#ff7675', '#fdcb6e', '#0984e3'],
      },
    ],
  };
}
```

```

> **\*\*Important Note\*\*:** Changing the chart data without creating a new data instance does not trigger change detection. In order to chart to redraw itself, a new data object needs to be created.

See the result:

![pie-chart](./images/pie-chart.png)

#### ## Examples

##### ### Doughnut

```

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-chart-demo',
  template: `
    <abp-chart
      type="doughnut"
      [data]="data"
      [options]="options"
      width="400px"
      height="400px"
    ></abp-chart>
  `,
})
export class ChartDemoComponent {
  data = {

```

```

    labels: ['Data 1', 'Data 2', 'Data 3'],
    datasets: [
      {
        label: 'Dataset 1',
        data: [40, 15, 45],
        backgroundColor: ['#a0e6c3', '#f0ea4c', '#5b9dc3'],
      },
    ],
  };
}

options = {
  plugins: {
    title: {
      display: true,
      text: 'Doughnut Chart',
      fontSize: 16,
    },
    legend: {
      position: 'bottom',
    },
  },
};

```

```

Result:

![Doughnut Chart](./images/doughnut-chart.png)

```

Bar

```ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-chart-demo',
  template: `
    <abp-chart
      type="bar"
      [data]="data"
      width="400px"
      height="400px"
    ></abp-chart>
  `,
})
export class ChartDemoComponent {
  data = {
    labels: ['January', 'February', 'March', 'April', 'May', 'June', 'July'],
    datasets: [
      {
        label: 'First dataset',
        backgroundColor: '#42A5F5',
      },
    ],
  };
}
```

```

```

 data: [65, 59, 80, 81, 56, 55, 40],
 },
 {
 label: 'Second dataset',
 backgroundColor: '#FFA726',
 data: [28, 48, 40, 19, 86, 27, 90],
 },
],
};

```

```

Result:

```

![Bar Chart](./images/bar-chart.png)

#### Radar

```ts
import { Component } from '@angular/core';

@Component({
 selector: 'app-chart-demo',
 template: `
 <abp-chart
 type="radar"
 [data]="data"
 width="400px"
 height="400px"
 ></abp-chart>

 <button class="btn btn-primary-outline mt-4" (click)="addDataset()">
 Add dataset
 </button>
 `,
})
export class ChartDemoComponent {
 data = {
 labels: [
 'January',
 'February',
 'March',
 'April',
 'May',
 'June',
 'July',
 'August',
 'September',
 'October',
 'November',
 'December',
],
 };
}
```

```

```

datasets: [
  {
    label: 'Dataset 1',
    backgroundColor: 'rgba(179, 181, 198, 0.2)',
    borderColor: 'rgba(179, 181, 198, 1)',
    data: [65, 59, 90, 81, 56, 55, 40, 35, 82, 51, 62, 95],
  },
  {
    label: 'Dataset 2',
    backgroundColor: 'rgba(255, 99, 132, 0.2)',
    borderColor: 'rgba(255, 99, 132, 1)',
    data: [28, 48, 40, 58, 96, 27, 100, 44, 85, 77, 71, 39],
  },
],
};

addDataset() {
  this.data = {
    ...this.data,
    datasets: [
      ...this.data.datasets,
      {
        label: 'Dataset 3',
        backgroundColor: 'rgba(54, 162, 235, 0.2)',
        borderColor: 'rgba(54, 162, 235, 1)',
        data: [90, 95, 98, 91, 99, 96, 89, 95, 98, 93, 92, 90],
      },
    ],
  };
}
```

```

Result:

![Bar Chart](./images/radar-chart.gif)

See the [`chart.js` samples](<https://www.chartjs.org/docs/latest/samples>) for more examples.

## API

#### `abp-chart`

##### Properties

| Name     | Description        |
|----------|--------------------|
| Type     | Default            |
| `[type]` | Type of the chart. |
| `string` | null               |

|                                          |                                                                  |  |
|------------------------------------------|------------------------------------------------------------------|--|
| <code>`[data]`</code>                    | Chart data to display                                            |  |
| <code>`any`</code>                       | null                                                             |  |
| <code>`[options]`</code>                 | Chart options to customize                                       |  |
| <code>`any`</code>                       | null                                                             |  |
| <code>`[plugins]`</code>                 | Chart plugins to customize behavior                              |  |
| <code>`any`</code>                       | null                                                             |  |
| <code>`[width]`</code>                   | Width of the chart                                               |  |
| <code>string</code>                      | null                                                             |  |
| <code>`[height]`</code>                  | Height of the chart                                              |  |
| <code>string</code>                      | null                                                             |  |
| <code>`[responsive]`</code>              | Whether the chart is responsive                                  |  |
| <code>boolean</code>                     | true                                                             |  |
| <code>`(dataSelect)`</code>              | A callback that executes when an element on the chart is clicked |  |
| <code>EventEmitter&lt;any&gt;</code>     | -                                                                |  |
| <code>`(initialized)`</code>             | A callback that executes when the chart is initialized           |  |
| <code>EventEmitter&lt;boolean&gt;</code> | -                                                                |  |

#### #### Methods

| Parameters                    | Description                                                         |
|-------------------------------|---------------------------------------------------------------------|
| <code>`refresh`</code>        | Redraws the chart                                                   |
| <code>`reinit`</code>         | Destroys the chart then creates it again                            |
| <code>`getBase64Image`</code> | Returns a base 64 encoded string of the chart in it's current state |
| <code>`generateLegend`</code> | Returns an HTML string of a legend for the chart                    |
| <code>`getCanvas`</code>      | Returns the canvas HTML element                                     |

#### 10.3.6.3 Card

##### # Card Component

The ABP Card Component is a wrapper component for the Bootstrap card class. It supports all the features that Bootstrap card component provides.

ABP Card Component has three main components, ``CardHeader``, ``CardBody`` and ``CardFooter``. These components have their own class and style inputs

| Component               | Selector                       | Input Properties                                                |
|-------------------------|--------------------------------|-----------------------------------------------------------------|
| <code>CardHeader</code> | <code>`abp-card-header`</code> | <code>`cardHeaderClass`</code> , <code>`cardHeaderStyle`</code> |
| <code>CardBody</code>   | <code>`abp-card-body`</code>   | <code>`cardBodyClass`</code> , <code>`cardBodyStyle`</code>     |
| <code>CardFooter</code> | <code>`abp-card-footer`</code> | <code>`cardFooterClass`</code> , <code>`cardFooterStyle`</code> |

In addition to these components, the Card component provides directives like `CardHeader`, `CardTitle`, `CardSubtitle`, `CardImgTop`.

| Directive    | Selector                                                        |
|--------------|-----------------------------------------------------------------|
| CardHeader   | `abp-card-header`, `[abp-card-header]`, `[abpCardHeader]`       |
| CardTitle    | `abp-card-title`, `[abp-card-title]`, `[abpCardTitle]`          |
| CardSubtitle | `abp-card-subtitle`, `[abp-card-subtitle]`, `[abpCardSubtitle]` |
| CardImgTop   | `abp-card-img-top`, `[abp-card-img-top]`, `[abpCardImgTop]`     |

### # Usage

ABP Card Component is a part of the `ThemeSharedModule` module. If you've imported that module into your module, you don't need to import it again. If not, first import it as shown below:

```
```ts
// my-feature.module.ts

import { ThemeSharedModule } from '@abp/ng.theme.shared';
import { CardDemoComponent } from './card-demo.component';

@NgModule({
  imports: [
    ThemeSharedModule,
    // ...
  ],
  declarations: [CardDemoComponent],
  // ...
})
export class MyFeatureModule {}
```

```

Then, the `abp-card` component can be used. See the examples below:

```
CardBody

```ts
// card-demo.component.ts

import { Component } from '@angular/core';

@Component({
  selector: 'app-card-demo',
  template: `
    <abp-card [cardStyle]="{{width: '18rem'}}>
      <abp-card-body>This is some text within a card body</abp-card-body>
    </abp-card>
  `,
})
```

```

```
export class CardDemoComponent { }
```

```

See the card body result below:

```
![abp-card-body](./images/abp-card-body.png)
```

```
## Titles, Text and Links
```

```
```ts
```

```
//card-demo.component.ts
import { Component } from '@angular/core';

@Component({
 selector: 'app-card-demo',
 template: `
 <abp-card [cardStyle]="{{width: '18rem'}}">
 <abp-card-body>
 <h5 abpCardTitle>Card Title</h5>
 <h6 abpCardSubtitle class="mb-2 text-muted">Card subtitle</h6>
 <p class="card-text">Some quick example text to build on the card title and make
 up the bulk of the card's content.</p>
 Card link
 Another link
 </abp-card-body>
 </abp-card>
 `,
})
export class CardDemoComponent { }
```

```

See the card title, text and link result below:

```
![abp-card-title-text-link](./images/abp-card-title-text-link.png)
```

```
## Images
```

```
```ts
```

```
//card-demo.component.ts
import { Component } from '@angular/core';

@Component({
 selector: 'app-card-demo',
 template: `
 <abp-card [cardStyle]="{{width:'18rem'}}">

 <abp-card-body>
 <p class="card-text" >Some quick example text to build on the card title and make
 up the bulk of the card's content.</p>
 </abp-card-body>
 </abp-card>
 `,
})
export class CardDemoComponent { }
```

```

```
)  
export class CardDemoComponent { }  
```
```

See the card image result below:

```
![abp-card-image-top](./images/abp-card-image.png)
```

```
List Groups
```

```
```ts
```

```
//card-demo.component.ts  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-card-demo',  
  template:  
    <abp-card [cardStyle]="{{width:'18rem'}}">  
      <ul class="list-group list-group-flush">  
        <li class="list-group-item">An item</li>  
        <li class="list-group-item">A second item</li>  
        <li class="list-group-item">A third item</li>  
      </ul>  
    </abp-card>  
  ,  
})  
export class CardDemoComponent { }  
```
```

See the group list result below:

```
![abp-card-list-group](./images/abp-card-list-group.png)
```

```
Kitchen Sink
```

```
```ts
```

```
//card-demo.component.ts  
import { Component } from '@angular/core';  
  
@Component({  
  selector: 'app-card-demo',  
  template:  
    <abp-card [cardStyle]="{{width:'18rem'}}">  
        
      <abp-card-body>  
        <h5 abpCardTitle>Card title</h5>  
        <p class="card-text">Some quick example text to build on the card title and make  
        up the bulk of the card's content.</p>  
      </abp-card-body>  
      <ul class="list-group list-group-flush">  
        <li class="list-group-item">An item</li>
```

```

        <li class="list-group-item">A second item</li>
        <li class="list-group-item">A third item</li>
    </ul>
    <abp-card-body>
        <a href="#" class="card-link">Card link</a>
        <a href="#" class="card-link">Another link</a>
    </abp-card-body>
</abp-card>
`,
})
```
export class CardDemoComponent { }
```

```

See kitchen sink result below:

```

![abp-card-kitchen-sink] ./images/abp-card-kitchen-sink.png

## Header and Footer

```ts

//card-demo.component.ts
import { Component } from '@angular/core';

@Component({
 selector: 'app-card-demo',
 template: `
 <abp-card class="text-center">
 <abp-card-header>Featured</abp-card-header>
 <abp-card-body>
 <h5 abpCardTitle>Special title treatment</h5>
 <p class="card-text">With supporting text below as a natural lead-in to additional
content.</p>
 Go somewhere
 </abp-card-body>
 <abp-card-footer class="text-muted">
 2 days ago
 </abp-card-footer>
 </abp-card>
 `,
})
```
export class CardDemoComponent { }
```

```

See the header and footer result below:

```

![abp-card-header-footer] ./images/abp-card-header-footer.png

```

## 10.4 React Native

### 10.4.1 Getting Started

```

```json

```

```
//[doc-params]
{
    "Tiered": ["No", "Yes"]
}
```
Getting Started with the React Native
```

ABP platform provide basic [[React Native](#)] (<https://reactnative.dev/>) startup template to develop mobile applications **\*\*integrated to your ABP based backends\*\***.

![React Native gif] (./images/react-native-introduction.gif)

## ## How to Prepare Development Environment

Please follow the steps below to prepare your development environment for React Native.

1. **\*\*Install Node.js:\*\*** Please visit [[Node.js downloads page](#)] (<https://nodejs.org/en/download/>) and download proper Node.js v16 or v18 installer for your OS. An alternative is to install [[NVM](#)] (<https://github.com/nvm-sh/nvm>) and use it to have multiple versions of Node.js in your operating system.
2. **\*\*[Optional] Install Yarn:\*\*** You may install Yarn v1 (not v2) following the instructions on [[the installation page](#)] (<https://classic.yarnpkg.com/en/docs/install>). Yarn v1 delivers an arguably better developer experience compared to npm v6 and below. You may skip this step and work with npm, which is built-in in Node.js, instead.
3. **\*\*[Optional] Install VS Code:\*\*** [[VS Code](#)] (<https://code.visualstudio.com/>) is a free, open-source IDE which works seamlessly with TypeScript. Although you can use any IDE including Visual Studio or Rider, VS Code will most likely deliver the best developer experience when it comes to React Native projects.
4. **\*\*Install an Emulator:\*\*** React Native applications need an Android emulator or an iOS simulator to run on your OS. See the [[Android Studio Emulator](#)] (<https://docs.expo.io/workflow/android-simulator/>) or [[iOS Simulator](#)] (<https://docs.expo.io/workflow/ios-simulator/>) on expo.io documentation to learn how to set up an emulator.

## ## How to Start a New React Native Project

You have multiple options to initiate a new React Native project that works with ABP:

### ### 1. Using ABP CLI

ABP CLI is probably the most convenient and flexible way to initiate an ABP solution with a React Native application. Simply [[install the ABP CLI](#)] (CLI.md) and run the following command in your terminal:

```
```shell
abp new MyCompanyName.MyProjectName -csf -u <angular or mvc> -m react-native
```
```

> To see further options in the CLI, please visit the [[CLI manual](#)] (CLI.md).

This command will prepare a solution with an **\*\*Angular\*\*** or an **\*\*MVC\*\*** (depends on your choice), a **\*\*.NET Core\*\***, and a **\*\*React Native\*\*** project in it.

## ### 2. Generating a CLI Command from Get Started Page

You can generate a CLI command on the [get started page of the abp.io website](<https://abp.io/get-started>). Then, use the command on your terminal to create a new [Startup Template](./Startup-Templates/Index.md).

### ## How to Configure & Run the Backend

➤ React Native application does not trust the auto-generated .NET HTTPS certificate. You should use **\*\*HTTP\*\*** during the development.

➤ When you are using OpenIddict, You should remove 'clientSecret' on Environment.js (if exists) and disable "HTTPS-only" settings. (OpenIddict has default since Version 6.0)

A React Native application running on an Android emulator or a physical phone **\*can not connect to the backend\*\*** on `localhost`. To fix this problem, it is necessary to run the backend application on your **\*\*local IP address\*\***.

```
 {{ if Tiered == "No" }}
![React Native host project local IP entry](images/rn-host-local-ip.png)

- Open the `appsettings.json` file in the `'.HttpApi.Host` folder. Replace the `localhost` address on the `SelfUrl` and `Authority` properties with your local IP address.
- Open the `launchSettings.json` file in the `'.HttpApi.Host/Properties` folder. Replace the `localhost` address on the `applicationUrl` properties with your local IP address.

 {{ else if Tiered == "Yes" }}

![React Native tiered project local IP entry](images/rn-tiered-local-ip.png)

- Open the `appsettings.json` file in the `'.AuthServer` folder. Replace the `localhost` address on the `SelfUrl` property with your local IP address.
- Open the `launchSettings.json` file in the `'.AuthServer/Properties` folder. Replace the `localhost` address on the `applicationUrl` properties with your local IP address.
- Open the `appsettings.json` file in the `'.HttpApi.Host` folder. Replace the `localhost` address on the `Authority` property with your local IP address.
- Open the `launchSettings.json` file in the `'.HttpApi.Host/Properties` folder. Replace the `localhost` address on the `applicationUrl` properties with your local IP address.

 {{ end }}
```

Run the backend application as described in the [getting started document](Getting-Started.md).

➤ You should turn off the "Https Restriction" if you're using OpenIddict as a central identity management solution. Because the IOS Simulator doesn't support self-signed certificates and OpenIddict is set to only work with HTTPS by default.

### ## How to disable the Https-only settings of OpenIddict

Open the `{{ if Tiered == "No" }}` `MyProjectNameHttpApiHostModule` `{{ else if Tiered == "Yes" }}` `MyProjectNameAuthServerModule` `{{ end }}` project and copy-paste the below code-block to the `PreConfigureServices` method:

```
```csharp
#if DEBUG
    PreConfigure<OpenIddictServerBuilder>(options =>
    {
        options.UseAspNetCore()
            .DisableTransportSecurityRequirement();
    });
#endif
```

```

## ## How to Configure & Run the React Native Application

1. Make sure the [database migration is complete](`./Getting-Started?UI=NG&DB=EF&Tiered=No#create-the-database`) and the [API is up and running](`./Getting-Started?UI=NG&DB=EF&Tiered=No#run-the-application`).
2. Open `react-native` folder and run `yarn` or `npm install` if you have not already.
3. Open the `Environment.js` in the `react-native` folder and replace the `localhost` address on the `apiUrl` and `issuer` properties with your local IP address as shown below:

![react native environment local IP](images/rn-environment-local-ip.png)

```
 {{ if Tiered == "Yes" }}

 > Make sure that `issuer` matches the running address of the `AuthServer` project,
 `apiUrl` matches the running address of the `HttpApi.Host` or `Web` project.

 {{else}}

 > Make sure that `issuer` and `apiUrl` matches the running address of the `HttpApi.Host`
 or `Web` project.

 {{ end }}

```

4. Run `yarn start` or `npm start`. Wait for the Expo CLI to print the options.  
  
 > The React Native application was generated with [Expo](https://expo.io/). Expo is a set of tools built around React Native to help you quickly start an app and, while it has many features.

![expo-cli-options](images/rn-options.png)

In the above image, you can start the application with an Android emulator, an iOS simulator or a physical phone by scanning the QR code with the [Expo Client](https://expo.io/tools#client) or choosing the option.

![React Native login screen on iPhone 11](images/rn-login-iphone.png)

Enter **\*\*admin\*\*** as the username and **\*\*1q2w3E\\*\*\*** as the password to login to the application.

The application is up and running. You can continue to develop your application based on this startup template.

## ## See Also

- [React Native project structure] (./Startup-Templates/Application#react-native)

## 10.5 Common

### 10.5.1.1 Themes

#### # The Official Themes

ABP Framework provides a complete UI theming system. While you can build your own themes, you can use the following pre-built themes freely in your applications.

#### ## The Basic Theme

The Basic Theme is a minimalist theme that doesn't add any styling on top of the plain [Bootstrap] (<https://getbootstrap.com/>) styles. You can take the Basic Theme as the base theme and build your own theme or styling on top of it. Here, a screenshot from the theme:

![basic-theme-application-layout] (./images/basic-theme-application-layout.png)

#### ### Documentation

- [Basic Theme - MVC UI] (./UI/AspNetCore/Basic-Theme.md)
- [Basic Theme - Blazor UI] (./UI/Blazor/Basic-Theme.md)
- [Basic Theme - Angular UI] (./UI/Angular/Basic-Theme.md)

#### ## The LeptonX Lite Theme

**LeptonX Lite** is the free version of the [LeptonX Theme] (<https://x.leptontheme.com/>), which is a part of the ABP Commercial. Here, a screenshot from the theme:

![LeptonX Lite application layout] (./images/leptonxlite-theme-application-layout.jpeg)

#### ### Documentation

- [LeptonX Lite - MVC UI] (LeptonXLite/AspNetCore.md)
- [LeptonX Lite - Blazor UI] (LeptonXLite/Blazor.md)
- [LeptonX Lite - Angular UI] (LeptonXLite/Angular.md)

## ## See Also

- \* [Theming - MVC UI] (./UI/AspNetCore/Theming.md)
- \* [Theming - Blazor UI] (./UI/Blazor/Theming.md)
- \* [Theming - Angular UI] (./UI/Angular/Theming.md)

### 10.5.1.2 Overriding the User Interface

## # Overriding the User Interface

You may want to override a page, a component, a JavaScript, CSS or an image file of your depended module. Overriding the UI completely depends on the UI framework you're using. Select the UI framework to continue:

- \* [ASP.NET Core (MVC / Razor Pages)] (UI/AspNetCore/Customization-User-Interface.md)
- \* [Angular] (UI/Angular/Customization-User-Interface.md)
- \* [Blazor] (UI/Blazor/Customization-Overriding-Components.md)

### 10.5.1.3 Utilities

#### 10.5.1.3.1 Linked List (Doubly)

##### # Linked List (Doubly)

The `@abp/utils` package provides a useful data structure known as a [doubly linked list] ([https://en.wikipedia.org/wiki/Doubly\\_linked\\_list](https://en.wikipedia.org/wiki/Doubly_linked_list)). It is available in both Angular (via an import) and MVC (via ``abp.utils.common`` global object).

Briefly, a doubly linked list is a series of records (a.k.a. nodes) which has information on the previous node, the next node, and its own value (or data).

##### ## Getting Started

To create a doubly linked list, all you have to do is to create a new instance of it:

In Angular:

```
```js
import { LinkedList } from '@abp/utils';

const list = new LinkedList();
```
```

In MVC:

```
```js
var list = new abp.utils.common.LinkedList();
```
```

The constructor does not get any parameters.

##### ## Usage

#### #### How to Add New Nodes

There are several methods to create new nodes in a linked list and all of them are separately available as well as revealed by `add` and `addMany` methods.

##### ##### addHead(value)

```
```js
addHead(value: T): ListNode<T>
````
```

Adds a node with given value as the first node in list:

```
```js
list.addHead('a');

// "a"

list.addHead('b');

// "b" <-> "a"

list.addHead('c');

// "c" <-> "b" <-> "a"
````
```

##### ##### addManyHead(values)

```
```js
addManyHead(values: T[]): ListNode<T>[]
````
```

Adds multiple nodes with given values as the first nodes in list:

```
```js
list.addManyHead(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

list.addManyHead(['x', 'y', 'z']);

// "x" <-> "y" <-> "z" <-> "a" <-> "b" <-> "c"
````
```

##### ##### addTail(value)

```
```js
addTail(value: T): ListNode<T>
````
```

Adds a node with given value as the last node in list:

```
```js
list.addTail('a');

// "a"

list.addTail('b');

// "a" <-> "b"

list.addTail('c');

// "a" <-> "b" <-> "c"
````
```

```
addManyTail(values)
```

```
```js
addManyTail(values: T[]): ListNode<T>[]
````
```

Adds multiple nodes with given values as the last nodes in list:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

list.addManyTail(['x', 'y', 'z']);

// "a" <-> "b" <-> "c" <-> "x" <-> "y" <-> "z"
````
```

```
addAfter(value, previousValue [, compareFn])
```

```
```js
addAfter(value: T, previousValue: T, compareFn?: ListComparisonFn<T>): ListNode<T>
````
```

Adds a node with given value after the first node that has the previous value:

```
```js
```

```

list.addTail('a');
list.addTail('b');
list.addTail('b');
list.addTail('c');

// "a" <-> "b" <-> "b" <-> "c"

list.addAfter('x', 'b');

// "a" <-> "b" <-> "x" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```

```js
list.addTail({ x: 1 });
list.addTail({ x: 2 });
list.addTail({ x: 3 });

// {"x":1} <-> {"x":2} <-> {"x":3}

list.addAfter(
  { x: 0 },
  2,
  (value, searchedValue) => value.x === searchedValue
);

// {"x":1} <-> {"x":2} <-> {"x":0} <-> {"x":3}
```

```

› The default compare function checks deep equality, so you will rarely need to pass that parameter.

```

addManyAfter(values, previousValue [, compareFn])

```js
addManyAfter(values: T[], previousValue: T, compareFn?: ListComparisonFn<T>):
ListNode<T>[]
```

```

Adds multiple nodes with given values after the first node that has the previous value:

```

```js
list.addManyTail(['a', 'b', 'b', 'c']);

// "a" <-> "b" <-> "b" <-> "c"

```

```
list.addManyAfter(['x', 'y'], 'b');

// "a" <-> "b" <-> "x" <-> "y" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addManyTail([{ x: 1 }, { x: 2 }, { x: 3 }]);

// {"x":1} <-> {"x":2} <-> {"x":3}

list.addManyAfter(
  [{ x: 4 }, { x: 5 }],
  2,
  (value, searchedValue) => value.x === searchedValue
);

// {"x":1} <-> {"x":2} <-> {"x":4} <-> {"x":5} <-> {"x":3}
```

```

➤ The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
addBefore(value, nextValue [, compareFn])

```js
addBefore(value: T, nextValue: T, compareFn?: ListComparisonFn<T>): ListNode<T>
```

```

Adds a node with given value before the first node that has the next value:

```
```js
list.addTail('a');
list.addTail('b');
list.addTail('b');
list.addTail('c');

// "a" <-> "b" <-> "b" <-> "c"

list.addBefore('x', 'b');

// "a" <-> "x" <-> "b" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addTail({ x: 1 });
list.addTail({ x: 2 });
list.addTail({ x: 3 });

// {"x":1} <-> {"x":2} <-> {"x":3}

list.addBefore(
  { x: 0 },
  2,
  (value, searchedValue) => value.x === searchedValue
);

// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":3}
```

```

› The default compare function checks deep equality, so you will rarely need to pass that parameter.

##### addManyBefore(values, nextValue [, compareFn])

```
```js
addManyBefore(values: T[], nextValue: T, compareFn?: ListComparisonFn<T>): ListNode<T>[]
```

```

Adds multiple nodes with given values before the first node that has the next value:

```
```js
list.addManyTail(['a', 'b', 'b', 'c']);

// "a" <-> "b" <-> "b" <-> "c"

list.addManyBefore(['x', 'y'], 'b');

// "a" <-> "x" <-> "y" <-> "b" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addManyTail([{ x: 1 }, { x: 2 }, { x: 3 }]);
```

```

```
// {"x":1} <-> {"x":2} <-> {"x":3}

list.addManyBefore(
 [{ x: 4 }, { x: 5 }],
 2,
 (value, searchedValue) => value.x === searchedValue
);

// {"x":1} <-> {"x":4} <-> {"x":5} <-> {"x":2} <-> {"x":3}
```

› The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
addByIndex(value, position)

```js
addByIndex(value: T, position: number): ListNode<T>
```

```

Adds a node with given value at the specified position in the list:

```
```js
list.addTail('a');
list.addTail('b');
list.addTail('c');

// "a" <-> "b" <-> "c"

list.addByIndex('x', 2);

// "a" <-> "b" <-> "x" <-> "c"
```

```

It works with negative index too:

```
```js
list.addTail('a');
list.addTail('b');
list.addTail('c');

// "a" <-> "b" <-> "c"

list.addByIndex('x', -1);

// "a" <-> "b" <-> "x" <-> "c"
```

```

```
```
```

```
##### addManyByIndex(values, position)

```js
addManyByIndex(values: T[], position: number): ListNode<T>[]
```

```

Adds multiple nodes with given values at the specified position in the list:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

list.addManyByIndex(['x', 'y'], 2);

// "a" <-> "b" <-> "x" <-> "y" <-> "c"
```

```

It works with negative index too:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

list.addManyByIndex(['x', 'y'], -1);

// "a" <-> "b" <-> "x" <-> "y" <-> "c"
```

```

```
##### add(value).head()

```js
add(value: T).head(): ListNode<T>
```

```

Adds a node with given value as the first node in list:

```
```js
list.add('a').head();

// "a"

list.add('b').head();

```

```
// "b" <-> "a"

list.add('c').head();

// "c" <-> "b" <-> "a"
```

➤ This is an alternative API for `addHead`.

```
add(value).tail()

```js  
add(value: T).tail(): ListNode<T>  
```
```

Adds a node with given value as the last node in list:

```
```js  
list.add('a').tail();  
  
// "a"  
  
list.add('b').tail();  
  
// "a" <-> "b"  
  
list.add('c').tail();  
  
// "a" <-> "b" <-> "c"  
```
```

➤ This is an alternative API for `addTail`.

```
add(value).after(previousValue [, compareFn])

```js  
add(value: T).after(previousValue: T, compareFn?: ListComparisonFn<T>): ListNode<T>  
```
```

Adds a node with given value after the first node that has the previous value:

```
```js  
list.add('a').tail();
```

```
list.add('b').tail();
list.add('b').tail();
list.add('c').tail();

// "a" <-> "b" <-> "b" <-> "c"

list.add('x').after('b');

// "a" <-> "b" <-> "x" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.add({ x: 1 }).tail();
list.add({ x: 2 }).tail();
list.add({ x: 3 }).tail();

// {"x":1} <-> {"x":2} <-> {"x":3}

list
  .add({ x: 0 })
  .after(2, (value, searchedValue) => value.x === searchedValue);

// {"x":1} <-> {"x":2} <-> {"x":0} <-> {"x":3}
```

```

> This is an alternative API for `addAfter`.  
>  
The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
add(value).before(nextValue [, compareFn])

```js
add(value: T).before(nextValue: T, compareFn?: ListComparisonFn<T>): ListNode<T>
```

```

Adds a node with given value before the first node that has the next value:

```
```js
list.add('a').tail();
list.add('b').tail();
list.add('b').tail();
list.add('c').tail();

```

```
// "a" <-> "b" <-> "b" <-> "c"  
  
list.add('x').before('b');  
  
// "a" <-> "x" <-> "b" <-> "b" <-> "c"  
```
```

You may pass a custom compare function to detect the searched value:

```
```js  
list.add({ x: 1 }).tail();  
list.add({ x: 2 }).tail();  
list.add({ x: 3 }).tail();  
  
// {"x":1} <-> {"x":2} <-> {"x":3}  
  
list  
.add({ x: 0 })  
.before(2, (value, searchedValue) => value.x === searchedValue);  
  
// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":3}  
```
```

➤ This is an alternative API for `addBefore`.  
➤  
➤ The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
add(value).byIndex(position)

```js  
add(value: T).byIndex(position: number): ListNode<T>  
```
```

Adds a node with given value at the specified position in the list:

```
```js  
list.add('a').tail();  
list.add('b').tail();  
list.add('c').tail();  
  
// "a" <-> "b" <-> "c"  
  
list.add('x').byIndex(2);  
  
// "a" <-> "b" <-> "x" <-> "c"
```

```
```
```

It works with negative index too:

```
```js
list.add('a').tail();
list.add('b').tail();
list.add('c').tail();

// "a" <-> "b" <-> "c"

list.add('x').byIndex(-1);

// "a" <-> "b" <-> "x" <-> "c"
```
```

› This is an alternative API for `addByIndex`.

```
addMany(values).head()
```

```
```js
addMany(values: T[]).head(): ListNode<T>[]
```
```

Adds multiple nodes with given values as the first nodes in list:

```
```js
list.addMany(['a', 'b', 'c']).head();

// "a" <-> "b" <-> "c"

list.addMany(['x', 'y', 'z']).head();

// "x" <-> "y" <-> "z" <-> "a" <-> "b" <-> "c"
```
```

› This is an alternative API for `addManyHead`.

```
addMany(values).tail()
```

```
```js
addMany(values: T[]).tail(): ListNode<T>[]
```
```

```
```
```

Adds multiple nodes with given values as the last nodes in list:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.addMany(['x', 'y', 'z']).tail();

// "a" <-> "b" <-> "c" <-> "x" <-> "y" <-> "z"
```
```

➤ This is an alternative API for ``addManyTail``.

```
#### addMany(values).after(previousValue [, compareFn])
```

```
```js
addMany(values: T[]).after(previousValue: T, compareFn?: ListComparisonFn<T>):
ListNode<T>[]
```
```

Adds multiple nodes with given values after the first node that has the previous value:

```
```js
list.addMany(['a', 'b', 'b', 'c']).tail();

// "a" <-> "b" <-> "b" <-> "c"

list.addMany(['x', 'y']).after('b');

// "a" <-> "b" <-> "x" <-> "y" <-> "b" <-> "c"
```
```

You may pass a custom compare function to detect the searched value:

```
```js
list.addMany([{ x: 1 }, { x: 2 }, { x: 3 }]).tail();

// {"x":1} <-> {"x":2} <-> {"x":3}

list
 .addMany([{ x: 4 }, { x: 5 }])
 .after(2, (value, searchedValue) => value.x === searchedValue);
```
```

```
// {"x":1} <-> {"x":2} <-> {"x":4} <-> {"x":5} <-> {"x":3}
```
> This is an alternative API for `addManyAfter`.
>
> The default compare function checks deep equality, so you will rarely need to pass that parameter.
```

```
addMany(values).before(nextValue [, compareFn])
```
js
addMany(values: T[]).before(nextValue: T, compareFn?: ListComparisonFn<T>): ListNode<T>[]
```

Adds multiple nodes with given values before the first node that has the next value:

```
```
js
list.addMany(['a', 'b', 'b', 'c']).tail();
// "a" <-> "b" <-> "b" <-> "c"

list.addMany(['x', 'y']).before('b');
// "a" <-> "x" <-> "y" <-> "b" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```
js
list.addMany([{ x: 1 }, { x: 2 }, { x: 3 }]).tail();
// {"x":1} <-> {"x":2} <-> {"x":3}

list
 .addMany([{ x: 4 }, { x: 5 }])
 .before(2, (value, searchedValue) => value.x === searchedValue);
// {"x":1} <-> {"x":4} <-> {"x":5} <-> {"x":2} <-> {"x":3}
```

```

```
> This is an alternative API for `addManyBefore`.
>
> The default compare function checks deep equality, so you will rarely need to pass that parameter.
```

```
#### addMany(values).byIndex(position)

```js
addMany(values: T[]).byIndex(position: number): ListNode<T>[]```
```

```

Adds multiple nodes with given values at the specified position in the list:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.addMany(['x', 'y']).byIndex(2);

// "a" <-> "b" <-> "x" <-> "y" <-> "c"
```
```

```

It works with negative index too:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.addMany(['x', 'y']).byIndex(-1);

// "a" <-> "b" <-> "x" <-> "y" <-> "c"
```
```

```

➤ This is an alternative API for `addManyByIndex`.

How to Remove Nodes

There are a few methods to remove nodes from a linked list and all of them are separately available as well as revealed from a `drop` method.

```
#### dropHead()

```js
dropHead(): ListNode<T> | undefined```
```

```

```
```
```

Removes the first node from the list:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropHead();

// "b" <-> "c"
```

```

```
dropManyHead(count)
```

```
```js
dropManyHead(count: number): ListNode<T>[]
```

```

Removes the first nodes from the list based on given count:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropManyHead(2);

// "c"
```

```

```
dropTail()
```

```
```js
dropTail(): ListNode<T> | undefined
```

```

Removes the last node from the list:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropTail();

// "a" <-> "b"
```

```

```
```
```

```
#### dropManyTail(count)

```js
dropManyTail(count: number): ListNode<T>[]
```
```

Removes the last nodes from the list based on given count:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropManyTail(2);

// "a"
```
```

```
#### dropByIndex(position)
```

```
```js
dropByIndex(position: number): ListNode<T> | undefined
```
```

Removes the node with the specified position from the list:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropByIndex(1);

// "a" <-> "c"
```
```

It works with negative index too:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropByIndex(-2);
```

```
// "a" <-> "c"
```
```
dropManyByIndex(count, position)
```
js
dropManyByIndex(count: number, position: number): ListNode<T>[]
```
```

Removes the nodes starting from the specified position from the list based on given count:

```
```js
list.addMany(['a', 'b', 'c', 'd']).tail();

// "a" <-> "b" <-> "c" <-> "d"

list.dropManyByIndex(2, 1);

// "a" <-> "d"
```
```

It works with negative index too:

```
```js
list.addMany(['a', 'b', 'c', 'd']).tail();

// "a" <-> "b" <-> "c" <-> "d

list.dropManyByIndex(2, -2);

// "a" <-> "d"
```

```

```
dropByValue(value [, compareFn])
```js  
dropByValue(value: T, compareFn?: ListComparisonFn<T>): ListNode<T> | undefined  
```
```

Removes the first node with given value from the list:

```
```js
list.addMany(['a', 'x', 'b', 'x', 'c']).tail();

// "a" <-> "x" <-> "b" <-> "x" <-> "c"
```

```
list.dropByValue('x');

// "a" <-> "b" <-> "x" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addMany([{ x: 1 }, { x: 0 }, { x: 2 }, { x: 0 }, { x: 3 }]).tail();

// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":0} <-> {"x":3}

list.dropByValue(0, (value, searchedValue) => value.x === searchedValue);

// {"x":1} <-> {"x":2} <-> {"x":0} <-> {"x":3}
```

```

➤ The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
dropByValueAll(value [, compareFn])

```js
dropByValueAll(value: T, compareFn?: ListComparisonFn<T>): ListNode<T>[]

```

```

Removes all nodes with given value from the list:

```
```js
list.addMany(['a', 'x', 'b', 'x', 'c']).tail();

// "a" <-> "x" <-> "b" <-> "x" <-> "c"

list.dropByValueAll('x');

// "a" <-> "b" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addMany([{ x: 1 }, { x: 0 }, { x: 2 }, { x: 0 }, { x: 3 }]).tail();

// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":0} <-> {"x":3}
```

```

```
list.dropByValue(0, (value, searchedValue) => value.x === searchedValue);
// {"x":1} <-> {"x":2} <-> {"x":3}
~~~
```

➤ The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
#### drop().head()  
~~~js  
drop().head(): ListNode<T> | undefined
~~~
```

Removes the first node in list:

```
~~~js  
list.addMany(['a', 'b', 'c']).tail();
// "a" <-> "b" <-> "c"

list.drop().head();
// "b" <-> "c"
~~~
```

➤ This is an alternative API for `dropHead`.

```
#### drop().tail()  
~~~js  
drop().tail(): ListNode<T> | undefined
~~~
```

Removes the last node in list:

```
~~~js  
list.addMany(['a', 'b', 'c']).tail();
// "a" <-> "b" <-> "c"

list.drop().tail();
```

```
// "a" <-> "b"
```
```

➤ This is an alternative API for `dropTail`.

```
#### drop().byIndex(position)  
  
```js  
drop().byIndex(position: number): ListNode<T> | undefined
```
```

Removes the node with the specified position from the list:

```
```js  
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.drop().byIndex(1);

// "a" <-> "c"
```
```

It works with negative index too:

```
```js  
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.drop().byIndex(-2);

// "a" <-> "c"
```
```

➤ This is an alternative API for `dropByIndex`.

```
#### drop().byValue(value [, compareFn])  
  
```js  
drop().byValue(value: T, compareFn?: ListComparisonFn<T>): ListNode<T> | undefined
```
```

Removes the first node with given value from the list:

```
```js
list.addMany(['a', 'x', 'b', 'x', 'c']).tail();

// "a" <-> "x" <-> "b" <-> "x" <-> "c"

list.drop().byValue('x');

// "a" <-> "b" <-> "x" <-> "c"
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addMany([{x: 1}, {x: 0}, {x: 2}, {x: 0}, {x: 3}]).tail();

// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":0} <-> {"x":3}

list
 .drop()
 .byValue(0, (value, searchedValue) => value.x === searchedValue);

// {"x":1} <-> {"x":2} <-> {"x":0} <-> {"x":3}
```

```

> This is an alternative API for `dropByValue`.
>
The default compare function checks deep equality, so you will rarely need to pass that parameter.

drop().byValueAll(value [, compareFn])

```
```js
drop().byValueAll(value: T, compareFn?: ListComparisonFn<T>): ListNode<T>[]```

```

Removes all nodes with given value from the list:

```
```js
list.addMany(['a', 'x', 'b', 'x', 'c']).tail();

// "a" <-> "x" <-> "b" <-> "x" <-> "c"

list.drop().byValueAll('x');
```

```

```
// "a" <-> "b" <-> "c"
````
```

You may pass a custom compare function to detect the searched value:

```
```js
list.addMany([{ x: 1 }, { x: 0 }, { x: 2 }, { x: 0 }, { x: 3 }]).tail();
// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":0} <-> {"x":3}

list
 .drop()
 .byValueAll(0, (value, searchedValue) => value.x === searchedValue);
// {"x":1} <-> {"x":2} <-> {"x":3}
````
```

> This is an alternative API for `dropByValueAll`.
>
> The default compare function checks deep equality, so you will rarely need to pass that parameter.

```
#### dropMany(count).head()
```

```
```js
dropMany(count: number).head(): ListNode<T>[]
````
```

Removes the first nodes from the list based on given count:

```
```js
list.addMany(['a', 'b', 'c']).tail();
// "a" <-> "b" <-> "c"

list.dropMany(2).head();

// "c"
````
```

> This is an alternative API for `dropManyHead`.

```
#### dropMany(count).tail()

```js
dropMany(count: number).tail(): ListNode<T>[]
```

```

Removes the last nodes from the list based on given count:

```
```js
list.addMany(['a', 'b', 'c']).tail();

// "a" <-> "b" <-> "c"

list.dropMany(2).tail();

// "a"
```

```

➤ This is an alternative API for `dropManyTail`.

```
#### dropMany(count).byIndex(position)

```js
dropMany(count: number).byIndex(position: number): ListNode<T>[]
```

```

Removes the nodes starting from the specified position from the list based on given count:

```
```js
list.addMany(['a', 'b', 'c', 'd']).tail();

// "a" <-> "b" <-> "c" <-> "d"

list.dropMany(2).byIndex(1);

// "a" <-> "d"
```

```

It works with negative index too:

```
```js
list.addMany(['a', 'b', 'c', 'd']).tail();

// "a" <-> "b" <-> "c" <-> "d"
```

```

```
list.dropMany(2).byIndex(-2);  
// "a" <-> "d"
```

➤ This is an alternative API for `dropManyByIndex`.

How to Find Nodes

There are a few methods to find specific nodes in a linked list.

head

```
```js  
head: ListNode<T> | undefined;
```
```

Refers to the first node in the list.

tail

```
```js  
tail: ListNode<T> | undefined;
```
```

Refers to the last node in the list.

length

```
```js  
length: number;
```
```

Is the total number of nodes in the list.

find(predicate)

```
```js  
find(predicate: ListIteratorFn<T>): ListNode<T> | undefined
```
```

Finds the first node from the list that matches the given predicate:

```
```js
list.addManyTail(['a', 'b', 'b', 'c']);

// "a" <-> "b" <-> "b" <-> "c"

var found = list.find(node => node.value === 'b');

/*
found.value === "b"
found.previous.value === "a"
found.next.value === "b"
*/
```

```

findIndex(predicate)

```
```js
findIndex(predicate: ListIteratorFn<T>): number
```

```

Finds the position of the first node from the list that matches the given predicate:

```
```js
list.addManyTail(['a', 'b', 'b', 'c']);

// "a" <-> "b" <-> "b" <-> "c"

var i0 = list.findIndex(node => node.next && node.next.value === 'b');
var i1 = list.findIndex(node => node.value === 'b');
var i2 = list.findIndex(node => node.previous && node.previous.value === 'b');
var i3 = list.findIndex(node => node.value === 'x');

/*
i0 === 0
i1 === 1
i2 === 2
i3 === -1
*/
```

```

get(position)

```
```js
get(position: number): ListNode<T> | undefined
```

```

Finds and returns the node with specific position in the list:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

var found = list.get(1);

/*
found.value === "b"
found.previous.value === "a"
found.next.value === "c"
*/
```

```

```
##### indexOf(value [, compareFn])
```

```
```js
indexOf(value: T, compareFn?: ListComparisonFn<T>): number
```

```

Finds the position of the first node from the list that has the given value:

```
```js
list.addManyTail(['a', 'b', 'b', 'c']);

// "a" <-> "b" <-> "b" <-> "c"

var i0 = list.indexOf('a');
var i1 = list.indexOf('b');
var i2 = list.indexOf('c');
var i3 = list.indexOf('x');

/*
i0 === 0
i1 === 1
i2 === 3
i3 === -1
*/
```

```

You may pass a custom compare function to detect the searched value:

```
```js
list.addManyTail([{ x: 1 }, { x: 0 }, { x: 2 }, { x: 0 }, { x: 3 }]);

// {"x":1} <-> {"x":0} <-> {"x":2} <-> {"x":0} <-> {"x":3}
```

```

```
var i0 = indexOf(1, (value, searchedValue) => value.x === searchedValue);
var i1 = indexOf(2, (value, searchedValue) => value.x === searchedValue);
var i2 = indexOf(3, (value, searchedValue) => value.x === searchedValue);
var i3 = indexOf(0, (value, searchedValue) => value.x === searchedValue);
var i4 = indexOf(4, (value, searchedValue) => value.x === searchedValue);

/*
i0 === 0
i1 === 2
i2 === 4
i3 === 1
i4 === -1
*/
```

```

➤ The default compare function checks deep equality, so you will rarely need to pass that parameter.

#### #### How to Check All Nodes

There are a few ways to iterate over or display a linked list.

#### ##### forEach(iteratorFn)

```
```js
forEach(iteratorFn: ListIteratorFn<T>): void
```

```

Runs a function on all nodes in a linked list from head to tail:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

list.forEach((node, index) => console.log(node.value + index));

// 'a0'
// 'b1'
// 'c2'
```

```

#### ##### \\*[Symbol.iterator]()

A linked list is iterable. In other words, you may use methods like `for...of` on it.

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

for(const node of list) { /* ES6 for...of statement */
  console.log(node.value);
}

// 'a'
// 'b'
// 'c'
```
```

#### toArray()

```
```js
toArray(): T[]
```
```

Converts a linked list to an array of values:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"

var arr = list.toArray();

/*
arr === ['a', 'b', 'c']
*/
```
```

#### toNodeArray()

```
```js
toNodeArray(): ListNode<T>[]
```
```

Converts a linked list to an array of nodes:

```
```js
list.addManyTail(['a', 'b', 'c']);

// "a" <-> "b" <-> "c"
```
```

```

var arr = list.toObjectArray();

/*
arr[0].value === 'a'
arr[1].value === 'a'
arr[2].value === 'a'
*/
```
##### toString([mapperFn])
```
js
toString(mapperFn: ListMapperFn<T> = JSON.stringify): string
```

```

Converts a linked list to a string representation of nodes and their relations:

```

```
js
list.addManyTail(['a', 2, 'c', { k: 4, v: 'd' }]);

// "a" <-> 2 <-> "c" <-> {"k":4,"v":"d"}

var str = list.toString();

/*
str === '"a" <-> 2 <-> "c" <-> {"k":4,"v":"d"}'
*/
```

```

You may pass a custom mapper function to map values before stringifying them:

```

```
js
list.addMany([{ x: 1 }, { x: 2 }, { x: 3 }, { x: 4 }, { x: 5 }]).tail0;

// {"x":1} <-> {"x":2} <-> {"x":3} <-> {"x":4} <-> {"x":5}

var str = list.toString(value => value.x);

/*
str === '1 <-> 2 <-> 3 <-> 4 <-> 5'
*/
```

```

```
## API

#### Classes

##### LinkedList

```js
export class LinkedList<T = any> {

 // properties and methods are explained above

}
```

```

```
##### ListNode

```js
export class ListNode<T = any> {
 next: ListNode | undefined;

 previous: ListNode | undefined;

 constructor(public readonly value: T) {}
}
```

```

`ListNode` is the node that is being stored in the `LinkedList` for every record.

- `value` is the value stored in the node and is passed through the constructor.
- `next` refers to the next node in the list.
- `previous` refers to the previous node in the list.

```
```js
list.addManyTail([0, 1, 2]);

console.log(
 list.head.value, // 0
 list.head.next.value, // 1
 list.head.next.next.value, // 2
 list.head.next.next.previous.value, // 1
 list.head.next.next.previous.previous.value, // 0
 list.tail.value, // 2
 list.tail.previous.value, // 1
 list.tail.previous.previous.value, // 0
 list.tail.previous.previous.next.value, // 1
 list.tail.previous.previous.next.next.value, // 2
);
```

```

```
### Types
```

```
#### ListMapperFn
```

```
```js
type ListMapperFn<T = any> = (value: T) => any;
```

```

This function is used in `toString` method to map the node values before generating a string representation of the list.

```
#### ListComparisonFn
```

```
```js
type ListComparisonFn<T = any> = (nodeValue: T, comparedValue: any) => boolean;
```

```

This function is used while adding, dropping, and finding nodes based on a comparison value.

```
#### ListIteratorFn
```

```
```js
type ListIteratorFn<T = any, R = boolean> = (
 node: ListNode<T>,
 index?: number,
 list?: LinkedList,
) => R;
```

```

This function is used while iterating over the list either to do something with each node or to find a node.

11 Data Access

11.1 Overall

```
# Data Access
```

ABP framework was designed as database agnostic. It can work any type of data source by the help of the [repository] (Repositories.md) and [unit of work] (Unit-Of-Work.md) abstractions. Currently, the following providers are implemented as official:

- * [Entity Framework Core] (Entity-Framework-Core.md) (works with [various DBMS and providers] (<https://docs.microsoft.com/en-us/ef/core/providers/>).)
- * [MongoDB] (MongoDB.md)
- * [Dapper] (Dapper.md)

See Also

- * [Connection Strings] (Connection-Strings.md)
- * [Data Seeding] (Data-Seeding.md)
- * [Data Filtering] (Data-Filtering.md)

11.2 Entity Framework Core

Entity Framework Core Integration

This document explains how to integrate EF Core as an ORM provider to ABP based applications and how to configure it.

Installation

`Volo.Abp.EntityFrameworkCore` is the main NuGet package for the EF Core integration. Install it to your project (for a layered application, to your data/infrastructure layer):

```
``` shell
abp add-package Volo.Abp.EntityFrameworkCore
```
```

> If you haven't done it yet, you first need to install the [ABP CLI] (CLI.md). For other installation options, see [the package description page] (<https://abp.io/package-detail/Volo.Abp.EntityFrameworkCore>).

>

> Note: Instead, you can directly download a [startup template] (<https://abp.io/Templates>) with EF Core pre-installed.

Database Management System Selection

Entity Framework Core supports various database management systems ([see all] (<https://docs.microsoft.com/en-us/ef/core/providers/>)). ABP framework and this document don't depend on any specific DBMS. If you are creating a [reusable application module] (Modules/Index.md), avoid to depend on a specific DBMS package. However, in a final application you eventually will select a DBMS.

> See [Switch to Another DBMS for Entity Framework Core] (Entity-Framework-Core-Other-DBMS.md) document to learn how to switch the DBMS.

Creating DbContext

Your `DbContext` class should be derived from `AbpDbContext<T>` as shown below:

```
```C#

```

```

using Microsoft.EntityFrameworkCore;
using Volo.Abp.EntityFrameworkCore;

namespace MyCompany.MyProject
{
 public class MyDbContext : AbpDbContext<MyDbContext>
 {
 //... your DbSet properties here

 public MyDbContext(DbContextOptions<MyDbContext> options)
 : base(options)
 {
 }

 }
}
```

```

About the EF Core Fluent Mapping

The [application startup template](Startup-Templates/Application.md) has been configured to use the [EF Core fluent configuration API](https://docs.microsoft.com/en-us/ef/core/modeling/) to map your entities to your database tables.

You can still use the ****data annotation attributes**** (like `#[Required]`) on the properties of your entity while the ABP documentation generally follows the ****fluent mapping API**** approach. It is up to you.

ABP Framework has some ****base entity classes**** and ****conventions**** (see the [entities document](Entities.md)) and it provides some useful ****extension methods**** to configure the properties inherited from the base entity classes.

ConfigureByConvention Method

`ConfigureByConvention()` is the main extension method that ****configures all the base properties**** and conventions for your entities. So, it is a ****best practice**** to call this method for all your entities, in your fluent mapping code.

****Example:**** Assume that you've a `Book` entity derived from `AggregateRoot<Guid>` base class:

```

```csharp
public class Book : AuditedAggregateRoot<Guid>
{
 public string Name { get; set; }
}
```

```

You can override the `OnModelCreating` method in your `DbContext` and configure the mapping as shown below:

```

```csharp
protected override void OnModelCreating(ModelBuilder builder)

```

```

{
 //Always call the base method
 base.OnModelCreating(builder);

 builder.Entity<Book>(b =>
 {
 b.ToTable("Books");

 //Configure the base properties
 b.ConfigureByConvention();

 //Configure other properties (if you are using the fluent API)
 b.Property(x => x.Name).IsRequired().HasMaxLength(128);
 });
}
```

```

* Calling `b.ConfigureByConvention()` is important here to properly ****configure the base properties****.

* You can configure the `Name` property here or you can use the ****data annotation attributes**** (see the [EF Core document](<https://docs.microsoft.com/en-us/ef/core/modeling/entity-properties>)).

> While there are many extension methods to configure your base properties, `ConfigureByConvention()` internally calls them if necessary. So, it is enough to call it.

Configure the Connection String Selection

If you have multiple databases in your application, you can configure the connection string name for your `DbContext` using the `[ConnectionStringName]` attribute. Example:

```

```csharp
[ConnectionStringName("MySecondConnString")]
public class MyDbContext : AbpDbContext<MyDbContext>
{
}
```

```

If you don't configure, the `Default` connection string is used. If you configure a specific connection string name, but not define this connection string name in the application configuration then it fallbacks to the `Default` connection string (see [the connection strings document](Connection-Strings.md) for more information).

AbpDbContextOptions

`AbpDbContextOptions` is used to configure the `DbContext` options. When you create a new solution with the ABP's application startup template, you will see a simple configuration (in the `EntityFrameworkCore` integration project's module class) as shown below:

```

```csharp
Configure<AbpDbContextOptions>(options =>
```

```

```
{
    options.UseSqlServer();
});
```

That configuration configures the default DBMS as SQL Server for all the `DbContext`'s of the application. That configuration was a shorthand notation and it can be done with the following code block:

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
 options.Configure(opts =>
 {
 opts.UseSqlServer();
 });
});
```

`options.Configure(...)` method has more options to configure. For example, you can set `DbContextOptions` (EF Core's native options) as shown below:

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
    options.Configure(opts =>
    {
        opts.DbContextOptions.UseQueryTrackingBehavior(QueryTrackingBehavior.NoTracking);
    });
});
```

If you have a single `DbContext` or you have multiple `DbContext`'s but want to use the same DBMS and configuration for all, you can leave it as is. However, if you need to configure a different DBMS or customize the configuration for a specific `DbContext`, you can specify it as shown below:

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
 // Default configuration for all DbContexts
 options.Configure(opts =>
 {
 opts.UseSqlServer();
 });

 // Customized configuration for a specific DbContext
 options.Configure<MyOtherDbContext>(opts =>
 {
 opts.UseMySQL();
 });
});
```

~~~~

> See [Switch to Another DBMS for Entity Framework Core] (Entity-Framework-Core-Other-DBMS.md) document to learn how to configure the DBMS.

### ## Registering DbContext To Dependency Injection

Use `AddAbpDbContext` method in your module to register your `DbContext` class for [dependency injection] (Dependency-Injection.md) system.

~~~~C#

```
using Microsoft.Extensions.DependencyInjection;
using Volo.Abp.EntityFrameworkCore;
using Volo.Abp.Modularity;

namespace MyCompany.MyProject
{
 [DependsOn(typeof(AbpEntityFrameworkCoreModule))]
 public class MyModule : AbpModule
 {
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 context.Services.AddAbpDbContext<MyDbContext>();

 //...
 }
 }
}~~~
```

### #### Add Default Repositories

ABP can automatically create default [generic repositories] (Repositories.md) for the entities in your DbContext. Just use `AddDefaultRepositories()` option on the registration:

~~~~C#

```
services.AddAbpDbContext<MyDbContext>(options =>
{
 options.AddDefaultRepositories();
});~~~
```

This will create a repository for each [aggregate root entity] (Entities.md) (classes derived from `AggregateRoot`) by default. If you want to create repositories for other entities too, then set `includeAllEntities` to `true`:

~~~~C#

```
services.AddAbpDbContext<MyDbContext>(options =>
{
 options.AddDefaultRepositories(includeAllEntities: true);
});~~~
```

~~~~

Then you can inject and use ` IRepository< TEntity, TPrimaryKey >` in your services. Assume that you have a ` Book` entity with ` Guid` primary key:

```
```csharp
public class Book : AggregateRoot<Guid>
{
    public string Name { get; set; }

    public BookType Type { get; set; }
}
````
```

(` BookType` is a simple ` enum` here and not important) And you want to create a new ` Book` entity in a [domain service] (Domain-Services.md) :

```
```csharp
public class BookManager : DomainService
{
    private readonly IRepository<Book, Guid> _bookRepository;

    //inject default repository to the constructor
    public BookManager(IRepository<Book, Guid> bookRepository)
    {
        _bookRepository = bookRepository;
    }

    public async Task<Book> CreateBook(string name, BookType type)
    {
        Check.NotNullOrWhiteSpace(name, nameof(name));

        var book = new Book
        {
            Id = GuidGenerator.Create(),
            Name = name,
            Type = type
        };

        //Use a standard repository method
        await _bookRepository.InsertAsync(book);

        return book;
    }
}
````
```

This sample uses ` InsertAsync` method to insert a new entity to the database.

#### Add Custom Repositories

Default generic repositories are powerful enough in most cases (since they implement ``IQueryable``). However, you may need to create a custom repository to add your own repository methods. Assume that you want to delete all books by type.

It's suggested to define an interface for your custom repository:

```
```csharp
public interface IBookRepository : IRepository<Book, Guid>
{
    Task DeleteBooksByType(BookType type);
}
```
```

You generally want to derive from the ``IRepository`` to inherit standard repository methods (while, you don't have to do). Repository interfaces are defined in the domain layer of a layered application. They are implemented in the data/infrastructure layer (`EntityFrameworkCore` project in a [startup template](<https://abp.io/Templates>)).

Example implementation of the ``IBookRepository`` interface:

```
```csharp
public class BookRepository
    : EfCoreRepository<BookStoreDbContext, Book, Guid>, IBookRepository
{
    public BookRepository(IDbContextProvider<BookStoreDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }

    public async Task DeleteBooksByType(BookType type)
    {
        var dbContext = await GetDbContextAsync();
        await dbContext.Database.ExecuteSqlRawAsync(
            $"DELETE FROM Books WHERE Type = {(int)type}"
        );
    }
}
```
```

Now, it's possible to [inject]([Dependency-Injection.md](#)) the ``IBookRepository`` and use the `DeleteBooksByType` method when needed.

#### #### Override the Default Generic Repository

Even if you create a custom repository, you can still inject the default generic repository (`IRepository<Book, Guid>` for this example). Default repository implementation will not use the class you have created.

If you want to replace default repository implementation with your custom repository, do it inside the ``AddAbpDbContext`` options:

```
```csharp
```

```

context.Services.AddAbpDbContext<BookStoreDbContext>(options =>
{
    options.AddDefaultRepositories();

    //Replaces IRepository<Book, Guid>
    options.AddRepository<Book, BookRepository>();
});
```

```

This is especially important when you want to **override a base repository method** to customize it. For instance, you may want to override `DeleteAsync` method to delete a specific entity in a more efficient way:

```

```csharp
public async override Task DeleteAsync(
    Guid id,
    bool autoSave = false,
    CancellationToken cancellationToken = default)
{
    //TODO: Custom implementation of the delete method
}
```

```

## ## Loading Related Entities

Assume that you've an `Order` with a collection of `OrderLine`s and the `OrderLine` has a navigation property to the `Order`:

```

```csharp
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Volo.Abp.Auditing;
using Volo.Abp.Domain.Entities;

namespace MyCrm
{
    public class Order : AggregateRoot<Guid>, IHasCreationTime
    {
        public Guid CustomerId { get; set; }
        public DateTime CreationTime { get; set; }

        public ICollection<OrderLine> Lines { get; set; } //Sub collection

        public Order()
        {
            Lines = new Collection<OrderLine>();
        }
    }

    public class OrderLine : Entity<Guid>
    {
```

```

```

 public Order Order { get; set; } //Navigation property
 public Guid OrderId { get; set; }

 public Guid ProductId { get; set; }
 public int Count { get; set; }
 public double UnitPrice { get; set; }
 }
}

````
```

And defined the database mapping as shown below:

```

```csharp
builder.Entity<Order>(b =>
{
 b.ToTable("Orders");
 b.ConfigureByConvention();

 //Define the relation
 b.HasMany(x => x.Lines)
 .WithOne(x => x.Order)
 .HasForeignKey(x => x.OrderId)
 .IsRequired();
});

builder.Entity<OrderLine>(b =>
{
 b.ToTable("OrderLines");
 b.ConfigureByConvention();
});
````
```

When you query an `Order`, you may want to ****include**** all the `OrderLine`s in a single query or you may want to ****load them later**** on demand.

> Actually these are not directly related to the ABP Framework. You can follow the [EF Core documentation](<https://docs.microsoft.com/en-us/ef/core/querying/related-data/>) to learn all the details. This section will cover some topics related to the ABP Framework.

Eager Loading / Load With Details

You have different options when you want to load the related entities while querying an entity.

Repository.WithDetails

` IRepository.WithDetailsAsync(...)` can be used to get an ` IQueryable<T>` by including one relation collection/property.

****Example: Get an order with lines****

```

```csharp
using System;
using System.Linq;
using System.Threading.Tasks;
using Volo.Abp.Domain.Repositories;
using Volo.Abp.Domain.Services;

namespace AbpDemo.Orders
{
 public class OrderManager : DomainService
 {
 private readonly IRepository<Order, Guid> _orderRepository;

 public OrderManager(IRepository<Order, Guid> orderRepository)
 {
 _orderRepository = orderRepository;
 }

 public async Task TestWithDetails(Guid id)
 {
 //Get a IQueryable<T> by including sub collections
 var queryable = await _orderRepository.WithDetailsAsync(x => x.Lines);

 //Apply additional LINQ extension methods
 var query = queryable.Where(x => x.Id == id);

 //Execute the query and get the result
 var order = await AsyncExecuter.FirstOrDefaultAsync(query);
 }
 }
}
```

```

> ```AsyncExecuter`` is used to execute async LINQ extensions without depending on the EF Core. If you add EF Core NuGet package reference to your project, then you can directly use `await query.FirstOrDefaultAsync()`. But, this time you depend on the EF Core in your domain layer. See the [repository document](Repositories.md) to learn more.

****Example: Get a list of orders with their lines****

```

```csharp
public async Task TestWithDetails()
{
 //Get a IQueryable<T> by including sub collections
 var queryable = await _orderRepository.WithDetailsAsync(x => x.Lines);

 //Execute the query and get the result
 var orders = await AsyncExecuter.ToListAsync(queryable);
}
```

```

> `WithDetailsAsync` method can get more than one expression parameter if you need to include more than one navigation property or collection.

DefaultWithDetailsFunc

If you don't pass any expression to the `WithDetailsAsync` method, then it includes all the details using the `DefaultWithDetailsFunc` option you provide.

You can configure `DefaultWithDetailsFunc` for an entity in the `ConfigureServices` method of your [module] (Module-Development-Basics.md) in your `EntityFrameworkCore` project.

Example: Include `Lines` while querying an `Order`

```
```csharp
Configure<AbpEntityOptions>(options =>
{
 options.Entity<Order>(orderOptions =>
 {
 orderOptions.DefaultWithDetailsFunc = query => query.Include(o => o.Lines);
 });
});
```

> You can fully use the EF Core API here since this is located in the EF Core integration project.

Then you can use the `WithDetails` without any parameter:

```
```csharp
public async Task TestWithDetails()
{
    //Get a IQueryable<T> by including all sub collections
    var queryable = await _orderRepository.WithDetailsAsync();

    //Execute the query and get the result
    var orders = await AsyncExecuter.ToListAsync(queryable);
}
```

`WithDetailsAsync()` executes the expression you've setup as the `DefaultWithDetailsFunc`.

Repository Get/Find Methods

Some of the standard [Repository] (Repositories.md) methods have optional `includeDetails` parameters;

- * `GetAsync` and `FindAsync` gets `includeDetails` with default value is `true`.
- * `GetListAsync` and `GetPagedListAsync` gets `includeDetails` with default value is `false`.

That means, the methods return a **single entity includes details** by default while list returning methods don't include details by default. You can explicitly pass ``includeDetails`` to change the behavior.

> These methods use the ``DefaultWithDetailsFunc`` option that is explained above.

****Example: Get an order with details****

```
~~~~csharp
public async Task TestWithDetails(Guid id)
{
    var order = await _orderRepository.GetAsync(id);
}
~~~~
```

****Example: Get an order without details****

```
~~~~csharp
public async Task TestWithoutDetails(Guid id)
{
    var order = await _orderRepository.GetAsync(id, includeDetails: false);
}
~~~~
```

****Example: Get list of entities with details****

```
~~~~csharp
public async Task TestWithDetails()
{
    var orders = await _orderRepository.GetListAsync(includeDetails: true);
}
~~~~
```

Alternatives

The repository pattern tries to encapsulate the EF Core, so your options are limited. If you need an advanced scenario, you can follow one of the options;

- * Create a custom repository method and use the complete EF Core API.
- * Reference to the ``Volo.Abp.EntityFrameworkCore`` package from your project. In this way, you can directly use ``Include`` and ``ThenInclude`` in your code.

See also [eager loading document] (<https://docs.microsoft.com/en-us/ef/core/querying/related-data/eager>) of the EF Core.

Explicit / Lazy Loading

If you don't include relations while querying an entity and later need to access to a navigation property or collection, you have different options.

EnsurePropertyLoadedAsync / EnsureCollectionLoadedAsync

Repositories provide `EnsurePropertyLoadedAsync` and `EnsureCollectionLoadedAsync` extension methods to **explicitly load** a navigation property or sub collection.

Example: Load Lines of an Order when needed

```
```csharp
public async Task TestWithDetails(Guid id)
{
 var order = await _orderRepository.GetAsync(id, includeDetails: false);
 //order.Lines is empty on this stage

 await _orderRepository.EnsureCollectionLoadedAsync(order, x => x.Lines);
 //order.Lines is filled now
}
````
```

`EnsurePropertyLoadedAsync` and `EnsureCollectionLoadedAsync` methods do nothing if the property or collection was already loaded. So, calling multiple times has no problem.

See also [explicit loading document] (<https://docs.microsoft.com/en-us/ef/core/querying/related-data/explicit>) of the EF Core.

Lazy Loading with Proxies

Explicit loading may not be possible in some cases, especially when you don't have a reference to the `Repository` or `DbContext`. Lazy Loading is a feature of the EF Core that loads the related properties / collections when you first access to it.

To enable lazy loading;

1. Install the [Microsoft.EntityFrameworkCore.Proxies] (<https://www.nuget.org/packages/Microsoft.EntityFrameworkCore.Proxies/>) package into your project (typically to the EF Core integration project)
2. Configure `UseLazyLoadingProxies` for your `DbContext` (in the `ConfigureServices` method of your module in your EF Core project). Example:

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
 options.PreConfigure<MyCrmDbContext>(opts =>
 {
 opts.DbContextOptions.UseLazyLoadingProxies(); //Enable lazy loading
 });

 options.UseSqlServer();
});
````
```

3. Make your navigation properties and collections `virtual`. Examples:

```
```csharp
```

```
public virtual ICollection<OrderLine> Lines { get; set; } //virtual collection
public virtual Order Order { get; set; } //virtual navigation property
````
```

Once you enable lazy loading and arrange your entities, you can freely access to the navigation properties and collections:

```
```csharp
public async Task TestWithDetails(Guid id)
{
 var order = await _orderRepository.GetAsync(id);
 //order.Lines is empty on this stage

 var lines = order.Lines;
 //order.Lines is filled (lazy loaded)
}
````
```

Whenever you access to a property/collection, EF Core automatically performs an additional query to load the property/collection from the database.

- Lazy loading should be carefully used since it may cause performance problems in some specific cases.

See also [lazy loading document] (<https://docs.microsoft.com/en-us/ef/core/querying/related-data/lazy>) of the EF Core.

Read-Only Repositories

ABP Framework provides read-only [repository] (Repositories.md) interfaces (``IReadOnlyRepository<...>`` or ``IReadOnlyBasicRepository<...>``) to explicitly indicate that your purpose is to query data, but not change it. If so, you can inject these interfaces into your services.

Entity Framework Core read-only repository implementation uses [EF Core's No-Tracking feature] (<https://learn.microsoft.com/en-us/ef/core/querying/tracking#no-tracking-queries>). That means the entities returned from the repository will not be tracked by the EF Core [change tracker] (<https://learn.microsoft.com/en-us/ef/core/change-tracking/>), because it is expected that you won't update entities queried from a read-only repository.

- This behavior works only if the repository object is injected with one of the read-only repository interfaces (``IReadOnlyRepository<...>`` or ``IReadOnlyBasicRepository<...>``). It won't work if you have injected a standard repository (e.g. ``IRepository<...>``) then casted it to a read-only repository interface.

Access to the EF Core API

In most cases, you want to hide EF Core APIs behind a repository (this is the main purpose of the repository pattern). However, if you want to access the `DbContext` instance over the repository, you can use `GetDbContext()` or `GetDbSet()` extension methods. Example:

```
```csharp
```

```

public async Task TestAsync()
{
 var dbContext = await _orderRepository.GetDbContextAsync();
 var dbSet = await _orderRepository.GetDbSetAsync();
 //var dbSet = dbContext.Set<Order>(); //Alternative, when you have the DbContext
}
```

```

* `GetDbContextAsync` returns a `DbContext` reference instead of `BookStoreDbContext`. You can cast it if you need. However, you don't need it in most cases.

> Important: You must reference to the `Volo.Abp.EntityFrameworkCore` package from the project you want to access to the `DbContext`. This breaks encapsulation, but this is what you want in that case.

Extra Properties & Object Extension Manager

Extra Properties system allows you to set/get dynamic properties to entities those implement the `IHasExtraProperties` interface. It is especially useful when you want to add custom properties to the entities defined in an [application module] (Modules/Index.md), when you use the module as package reference.

By default, all the extra properties of an entity are stored as a single `JSON` object in the database.

Entity extension system allows you to store desired extra properties in separate fields in the related database table. For more information about the extra properties & the entity extension system, see the following documents:

- * [Customizing the Application Modules: Extending Entities] (Customizing-Application-Modules-Extending-Entities.md)
- * [Entities] (Entities.md)

This section only explains the EF Core related usage of the `ObjectExtensionManager`.

ObjectExtensionManager.Instance

`ObjectExtensionManager` implements the singleton pattern, so you need to use the static `ObjectExtensionManager.Instance` to perform all the operations.

MapEfCoreProperty

`MapEfCoreProperty` is a shortcut extension method to define an extension property for an entity and map to the database.

****Example**:** Add `Title` property (database field) to the `IdentityRole` entity:

```

````csharp
ObjectExtensionManager.Instance
 .MapEfCoreProperty<IdentityRole, string>(
 "Title",
 (entityBuilder, propertyBuilder) =>
```

```

```

    {
        propertyBuilder.HasMaxLength(64);
    }
};

#### MapEfCoreEntity

`MapEfCoreEntity` is a shortcut extension method to configure the `Entity`.
```

****Example**:** Set the max length of `Name` to the `IdentityRole` entity:

```
```csharp
ObjectExtensionManager.Instance
 .MapEfCoreEntity<IdentityRole>(builder =>
{
 builder.As<EntityTypeBuilder<IdentityRole>>().Property(x =>
x.Name).HasMaxLength(200);
});
```

```

MapEfCoreDbContext

`MapEfCoreDbContext` is a shortcut extension method to configure the `DbContext`.

****Example**:** Set the max length of `Name` to the `IdentityRole` entity of `IdentityDbContext`:

```
```csharp
ObjectExtensionManager.Instance.MapEfCoreDbContext<IdentityDbContext>(b =>
{
 b.Entity<IdentityRole>().Property(x => x.Name).HasMaxLength(200);
});
```

```

If the related module has implemented this feature(explained below), then the new property is added to the model or the DbContext/Entity configure changed. Then you need to run the standard `Add-Migration` and `Update-Database` commands to update your database to add the new field.

> The `MapEfCoreProperty`, `MapEfCoreEntity` and `MapEfCoreDbContext` methods must be called before using the related `DbContext`. It is a static method. The best way is to use it in your application as earlier as possible. The application startup template has a `YourProjectNameEfCoreEntityExtensionMappings` class that is safe to use this method inside.

ConfigureEfCoreEntity, ApplyObjectExtensionMappings and TryConfigureObjectExtensions

If you are building a reusable module and want to allow application developers to add properties to your entities, you can use the `ConfigureEfCoreEntity`, `ApplyObjectExtensionMappings` and `TryConfigureObjectExtensions` extension methods in your entity mapping.

```

**Example**:
```csharp
public static class QADbContextModelCreatingExtensions
{
 public static void ConfigureQA(
 this ModelBuilder builder,
 Action<QAModelBuilderConfigurationOptions> optionsAction = null)
 {
 Check.NotNull(builder, nameof(builder));

 var options = new QAModelBuilderConfigurationOptions(
 QADatabaseDbProperties.DbTablePrefix,
 QADatabaseDbProperties.DbSchema
);

 optionsAction?.Invoke(options);

 builder.Entity<QA_Question>(b =>
 {
 b.ToTable(options.TablePrefix + "Questions", options.Schema);
 b.ConfigureByConvention();
 //...

 //Call this in the end of buildAction.
 b.ApplyObjectExtensionMappings();
 });
 }

 //...
}

//Call this in the end of ConfigureQA.
builder.TryConfigureObjectExtensions<QADbContext>();
}
```

```

> If you call `ConfigureByConvention()` extension method (like `b.ConfigureByConvention()` for this example), ABP Framework internally calls the `ConfigureObjectExtensions` and `ConfigureEfCoreEntity` methods. It is a ****best practice**** to use the `ConfigureByConvention()` method since it also configures database mapping for base properties by convention.

See the "**ConfigureByConvention Method**" section above for more information.

Advanced Topics

Controlling the Multi-Tenancy

If your solution is [multi-tenant] (Multi-Tenancy.md), tenants may have ****separate databases****, you have ****multiple**** `DbContext` classes in your solution and some of your `DbContext` classes should be usable ****only from the host side****, it is suggested to add `[IgnoreMultiTenancy]` attribute on your `DbContext` class. In this case, ABP guarantees

that the related `DbContext` always uses the host [connection string] (Connection-Strings.md), even if you are in a tenant context.

****Example:****

```
```csharp
[IgnoreMultiTenancy]
public class MyDbContext : AbpDbContext<MyDbContext>
{
 ...
}
````
```

Do not use the `[IgnoreMultiTenancy]` attribute if any one of your entities in your `DbContext` can be persisted in a tenant database.

> When you use repositories, ABP already uses the host database for the entities don't implement the `IMultiTenant` interface. So, most of time you don't need to `[IgnoreMultiTenancy]` attribute if you are using the repositories to work with the database.

Set Default Repository Classes

Default generic repositories are implemented by `EfCoreRepository` class by default. You can create your own implementation and use it for all the default repository implementations.

First, define your default repository classes like that:

```
```csharp
public class MyRepositoryBase<TEntity>
 : EfCoreRepository<BookStoreDbContext, TEntity>
 where TEntity : class, IEntity
{
 public MyRepositoryBase(IDbContextProvider<BookStoreDbContext> dbContextProvider)
 : base(dbContextProvider)
 {
 }
}

public class MyRepositoryBase<TEntity, TKey>
 : EfCoreRepository<BookStoreDbContext, TEntity, TKey>
 where TEntity : class, IEntity<TKey>
{
 public MyRepositoryBase(IDbContextProvider<BookStoreDbContext> dbContextProvider)
 : base(dbContextProvider)
 {
 }
}
````
```

First one is for [entities with composite keys](Entities.md), second one is for entities with single primary key.

It's suggested to inherit from the `EfCoreRepository` class and override methods if needed. Otherwise, you will have to implement all the standard repository methods manually.

Now, you can use `SetDefaultRepositoryClasses` option:

```
```csharp
context.Services.AddAbpDbContext<BookStoreDbContext>(options =>
{
 options.SetDefaultRepositoryClasses(
 typeof(MyRepositoryBase<, >),
 typeof(MyRepositoryBase<>)
);

 //...
});```

```

### ### Set Base DbContext Class or Interface for Default Repositories

If your DbContext inherits from another DbContext or implements an interface, you can use that base class or interface as DbContext for default repositories. Example:

```
```csharp
public interface IBookStoreDbContext : IEfCoreDbContext
{
    DbSet<Book> Books { get; }
}
```
```

`IBookStoreDbContext` is implemented by the `BookStoreDbContext` class. Then you can use generic overload of the `AddDefaultRepositories`:

```
```csharp
context.Services.AddAbpDbContext<BookStoreDbContext>(options =>
{
    options.AddDefaultRepositories<IBookStoreDbContext>();
    //...
});```

```

Now, your custom `BookRepository` can also use the `IBookStoreDbContext` interface:

```
```csharp
public class BookRepository : EfCoreRepository<IBookStoreDbContext, Book, Guid>,
IBookRepository
{
 //...
}
```

~~~~

One advantage of using an interface for a DbContext is then it will be replaceable by another implementation.

#### #### Replace Other DbContextes

Once you properly define and use an interface for DbContext, then any other implementation can use the following ways to replace it:

##### ##### ReplaceDbContext Attribute

```
```csharp
[ReplaceDbContext(typeof(IBookStoreDbContext))]
public class OtherDbContext : AbpDbContext<OtherDbContext>, IBookStoreDbContext
{
    //...
}
````
```

##### ##### ReplaceDbContext Option

```
```csharp
context.Services.AddAbpDbContext<OtherDbContext>(options =>
{
    //...
    options.ReplaceDbContext<IBookStoreDbContext>();
});
````
```

In this example, `OtherDbContext` implements `IBookStoreDbContext`. This feature allows you to have multiple DbContext (one per module) on development, but single DbContext (implements all interfaces of all DbContexts) on runtime.

##### ##### Replacing with Multi-Tenancy

It is also possible to replace a DbContext based on the [multi-tenancy](Multi-Tenancy.md) side. `ReplaceDbContext` attribute and `ReplaceDbContext` method can get a `MultiTenancySides` option with a default value of `MultiTenancySides.Both`.

**\*\*Example:\*\*** Replace DbContext only for tenants, using the `ReplaceDbContext` attribute

```
```csharp
[ReplaceDbContext(typeof(IBookStoreDbContext), MultiTenancySides.Tenant)]
````
```

**\*\*Example:\*\*** Replace DbContext only for the host side, using the `ReplaceDbContext` method

```
```csharp
options.ReplaceDbContext<IBookStoreDbContext>(MultiTenancySides.Host);
````
```

### ### Split Queries

ABP enables [split queries](<https://docs.microsoft.com/en-us/ef/core/querying/single-split-queries>) globally by default for better performance. You can change it as needed.

#### \*\*Example\*\*

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
    options.UseSqlServer(optionsBuilder =>
    {
        optionsBuilder.UseQuerySplittingBehavior(QuerySplittingBehavior.SingleQuery);
    });
});
```

Customize Bulk Operations

If you have better logic or using an external library for bulk operations, you can override the logic via implementing `IEfCoreBulkOperationProvider`.

- You may use example template below:

```
```csharp
public class MyCustomEfCoreBulkOperationProvider
 : IEfCoreBulkOperationProvider, ITransientDependency
{
 public async Task DeleteManyAsync<TDbContext, TEntity>(
 IEfCoreRepository<TEntity> repository,
 IEnumerable<TEntity> entities,
 bool autoSave,
 CancellationToken cancellationToken)
 where TDbContext : IEfCoreDbContext
 where TEntity : class, IEntity
 {
 // Your logic here.
 }

 public async Task InsertManyAsync<TDbContext, TEntity>(
 IEfCoreRepository<TEntity> repository,
 IEnumerable<TEntity> entities,
 bool autoSave,
 CancellationToken cancellationToken)
 where TDbContext : IEfCoreDbContext
 where TEntity : class, IEntity
 {
 // Your logic here.
 }

 public async Task UpdateManyAsync<TDbContext, TEntity>(
 IEfCoreRepository<TEntity> repository,
```

```

IEnumerable< TEntity > entities,
bool autoSave,
CancellationToken cancellationToken)
where TDbContext : IEfCoreDbContext
where TEntity : class, IEntity
{
 // Your logic here.
}
```

```

See Also

- * [Entities] (Entities.md)
- * [Repositories] (Repositories.md)

11.2.1 Database Migrations

EF Core Database Migrations

This document begins by ****introducing the default structure**** provided by [\[the application startup template\]](#) (Startup-Templates/Application.md) and ****discusses various scenarios**** you may want to implement for your own application.

> This document is for who want to fully understand and customize the database structure comes with [\[the application startup template\]](#) (Startup-Templates/Application.md). If you simply want to create entities and manage your code first migrations, just follow [\[the startup tutorials\]](#) (Tutorials/Part-1.md).

Source Code

You can find the source code of the example project referenced by this document [\[here\]](#) (<https://github.com/abpframework/abp-samples/tree/master/EfCoreMigrationDemo>). However, you need to read and understand this document in order to understand the example project's source code.

About the EF Core Code First Migrations

Entity Framework Core provides an easy to use and powerful [\[database migration system\]](#) (<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). ABP Framework [\[startup templates\]](#) (Startup-Templates/Index.md) take the advantage of this system to allow you to develop your application in a standard way.

However, EF Core migration system is ****not so good in a modular environment**** where each module maintains its ****own database schema**** while two or more modules may ****share a single database**** in practical.

Since ABP Framework cares about modularity in all aspects, it provides a ****solution**** to this problem. It is important to understand this solution if you need to ****customize your database structure****.

› See [EF Core's own documentation] (<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>) to fully learn the EF Core Code First Migrations and why you need to such a system.

The Default Solution & Database Configuration

When you [create a new web application] (<https://abp.io/get-started>) (with EF Core, which is the default database provider), your solution structure will be similar to the picture below:

![bookstore-visual-studio-solution-v3] (images/bookstore-visual-studio-solution-v3.png)

Actual solution structure may be a bit different based on your preferences, but the database part will be same.

› This document will use the `Acme.BookStore` example project name to refer the projects and classes. You need to find the corresponding class/project in your solution.

The Database Structure

The startup template has some [application modules] (Modules/Index.md) pre-installed. Each layer of the solution has corresponding module ****package references****. So, the `EntityFrameworkCore` project has the NuGet references for the `EntityFrameworkCore` packages of the used modules:

![bookstore-efcore-dependencies] (images/bookstore-efcore-dependencies.png)

In this way, you collect all the ****EF Core dependencies**** under the `EntityFrameworkCore` project.

› In addition to the module references, it references to the `Volo.Abp.EntityFrameworkCore.SqlServer` package since the startup template is pre-configured for the ****SQL Server****. See the documentation if you want to [switch to another DBMS] (Entity-Framework-Core-Other-DBMS.md).

While every module has its own `DbContext` class by design and can use its ****own physical database****, the solution is configured to use a ****single shared database**** as shown in the figure below:

![single-database-usage] (images/single-database-usage.png)

This is ****the simplest configuration**** and suitable for most of the applications. `appsettings.json` file has a ****single connection string****, named `Default`:

```
```json
"ConnectionStrings": {
 "Default": "..."
}
```
```

So, you have a **single database schema** which contains all the tables of the modules **sharing** this database.

ABP Framework's [connection string] (Connection-Strings.md) system allows you to easily **set a different connection string** for a desired module:

```
```json
".ConnectionStrings": {
 "Default": "...",
 "AbpAuditLogging": "..."
}
````
```

The example configuration tells to the ABP Framework to use the second connection string for the [Audit Logging module] (Modules/Audit-Logging.md) (if you don't specify connection string for a module, it uses the `Default` connection string).

However, this can work only if the audit log database with the given connection string is available. So, you need to create the second database, create audit log tables inside it and maintain the database tables. No problem if you manually do all these. However, the recommended approach is the code first migrations. One of the main purposes of this document is to guide you on such **database separation** scenarios.

Module Tables

Every module uses its **own databases tables**. For example, the [Identity Module] (Modules/Identity.md) has some tables to manage the users and roles in the system.

Table Prefixes

Since it is allowed to share a single database by all modules (it is the default configuration), a module typically uses a **table name prefix** to group its own tables.

The fundamental modules, like [Identity] (Modules/Identity.md), [Tenant Management] (Modules/Tenant-Management.md) and [Audit Logs] (Modules/Audit-Logging.md), use the `Abp` prefix, while some other modules use their own prefixes. [Identity Server] (Modules/IdentityServer.md) module uses the `IdentityServer` prefix for example.

If you want, you can **change the database table name prefix** for a module for your application. Example:

```
```csharp
Volo.Abp.IdentityServer.AbpNetCoreIdentityServerDbProperties.DbTablePrefix = "Ids";
````
```

This code changes the prefix of the [Identity Server] (Modules/IdentityServer.md) module. Write this code **at the very beginning** in your application.

> Every module also defines `DbSchema` property (near to `DbTablePrefix`), so you can set it for the databases support the schema usage.

.EntityFrameworkCore Project

The solution contains a project, which's name ends with ` `. EntityFrameworkCore` . This project has the ` `DbContext` class (` BookStoreDbContext` for this sample) of your application.

****Every module uses its own ` `DbContext` class**** to access to the database. Likewise, your application has its own ` `DbContext` . You typically use this ` `DbContext` in your application code (in your [repositories] (Repositories.md) if you follow the best practices and hide your data access code behind the repositories). It is almost an empty ` `DbContext` since your application don't have any entities at the beginning:

```
```csharp
[ReplaceDbContext(typeof(IIdentityDbContext))]
[ReplaceDbContext(typeof(ITenantManagementDbContext))]
[ConnectionStringName("Default")]
public class BookStoreDbContext :
 AbpDbContext<BookStoreDbContext>,
 IIdentityDbContext,
 ITenantManagementDbContext
{
 /* Add DbSet properties for your Aggregate Roots / Entities here. */

 /* DbSet for entities from the replaced DbContexts */

 public BookStoreDbContext(DbContextOptions<BookStoreDbContext> options)
 : base(options)
 {
 }

 protected override void OnModelCreating(ModelBuilder builder)
 {
 base.OnModelCreating(builder);

 /* Include modules to your migration db context */
 builder.ConfigurePermissionManagement();
 builder.ConfigureSettingManagement();
 builder.ConfigureBackgroundJobs();
 builder.ConfigureAuditLogging();
 builder.ConfigureIdentity();
 builder.ConfigureIdentityServer();
 builder.ConfigureFeatureManagement();
 builder.ConfigureTenantManagement();

 /* Configure your own tables/entities here. Example: */
 //builder.Entity<YourEntity>(b =>
 //{
 // b.ToTable("YourEntities");
 // b.ConfigureByConvention(); //auto configure for the base properties
 // // ...
 //});
 }
}
```

```
}
```

This `DbContext` class needs some explanations:

- \* It defines `[ReplaceDbContext]` attributes for `IIdentityDbContext` and `ITenantManagementDbContext` those replaces Identity and Tenant Management module's `DbContext`s by your `DbContext` on runtime. This allows us to easily perform LINQ queries by joining your entities with the entities (over the repositories) coming from those modules.
- \* It defines a `[ConnectionStringName]` attribute which tells ABP to always use the `Default` connection string for this `DbContext`.
- \* It inherits from the `AbpDbContext<T>` instead of the standard `DbContext` class. You can see the [EF Core integration](Entity-Framework-Core.md) document for more. For now, know that the `AbpDbContext<T>` base class implements some conventions of the ABP Framework to automate some common tasks for you.
- \* It declares `DbSet` properties for entities from the replaced `DbContext`s (by implementing the corresponding interfaces). These `DbSet` properties are not shown above (for the sake of brevity), but you can find in your application's code in a `region`.
- \* The constructor takes a `DbContextOptions<T>` instance.
- \* It overrides the `OnModelCreating` method to define the EF Core mappings.
  - \* It first calls the the `base.OnModelCreating` method to let the ABP Framework to implement the base mappings for us.
  - \* It then calls some `builder.ConfigureXXX()` methods for the used modules. This makes possible to add database mappings for these modules to this `DbContext`, so it creates the database tables of the modules when we add a new EF Core database migration.
  - \* You can configure the mappings for your own entities as commented in the example code. At this point, you can also change mappings for the modules you are using.

### ### Discussion of an Alternative Scenario: Every Module Manages Its Own Migration Path

As mentioned before, in the `EntityFrameworkCore` project, we merge all the database mappings of all the modules (plus your application's mappings) to create a unified migration path.

An alternative approach would be to allow each module to have its own migrations to maintain its database tables. While it seems more modular in the beginning, it has some drawbacks in practical:

- \* **EF Core migration system depends on the DBMS provider.** For example, if a module has created migrations for SQL Server, then you can not use this migration code for MySQL. It is not practical for a module to maintain migrations for all available DBMS providers. Leaving the migration to the application code (as explained in this document) allows you to **choose the DBMS in the application** code. If you can depend on a specific DBMS in your module, that's not an issue for you, however all pre-built ABP modules are DBMS agnostic.
- \* It would be harder to **customize/enhance** the mapping and the resulting migration code, in the final application.
- \* It would be harder to track and **apply changes** to database when you use multiple modules.

### ## Using Multiple Databases

The default startup template is organized to use a **single database** used by all the modules and by your application. However, the ABP Framework and all the pre-built modules are designed so that **they can use multiple databases**. Each module can use its own database or you can group modules into a few databases.

This section will explain how to move Audit Logging, Setting Management and Permission Management module tables to a **second database** while the remaining modules continue to use the main ("Default") database.

The resulting structure will be like the figure below:

![single-database-usage](images/multiple-database-usage.png)

#### ### Change the Connection Strings Section

First step is to change the connection string section inside all the `appsettings.json` files. Initially, it is like that:

```
```json
"ConnectionStrings": {
    "Default": "Server=(LocalDb)\\MSSQLLocalDB;Database=BookStore;Trusted_Connection=True"
}
```
```

Change it as shown below:

```
```json
"ConnectionStrings": {
    "Default": "Server=(LocalDb)\\MSSQLLocalDB;Database=BookStore;Trusted_Connection=True",
    "AbpPermissionManagement": "Server=(LocalDb)\\MSSQLLocalDB;Database=BookStore_SecondDb;Trusted_Connection=True",
    "AbpSettingManagement": "Server=(LocalDb)\\MSSQLLocalDB;Database=BookStore_SecondDb;Trusted_Connection=True",
    "AbpAuditLogging": "Server=(LocalDb)\\MSSQLLocalDB;Database=BookStore_SecondDb;Trusted_Connection=True"
}
```
```

Added **three more connection strings** for the related module to target the `BookStore\_SecondDb` database (they are all the same). For example, `AbpPermissionManagement` is the connection string name used by the permission management module.

The `AbpPermissionManagement` is a constant [defined](https://github.com/abpframework/abp/blob/97eaa6ff5a044f503465455c86332e5a277b077a/modules/permission-management/src/Volo.Abp.PermissionManagement.Domain/Volo/Abp/PermissionManagement/AbpPermissionManagementDbProperties.cs#L11) by the permission management module. ABP Framework [connection string selection system](Connection-Strings.md) selects this connection string for the permission management module if you define. If you don't define, it fallbacks to the `Default` connection string.

#### #### Create a Second DbContext

Defining the connection strings as explained above is enough **\*\*on runtime\*\***. However, `BookStore\_SecondDb` database doesn't exist yet. You need to create the database and the tables for the related modules.

Just like the main database, we want to use the EF Core Code First migration system to create and maintain the second database. So, create a new `DbContext` class inside the `BookStore.EntityFrameworkCore` project:

```
```csharp
using Microsoft.EntityFrameworkCore;
using Volo.Abp.AuditLogging.EntityFrameworkCore;
using Volo.Abp.Data;
using Volo.Abp.EntityFrameworkCore;
using Volo.Abp.PermissionManagement.EntityFrameworkCore;
using Volo.Abp.SettingManagement.EntityFrameworkCore;

namespace BookStore.EntityFrameworkCore
{
    [ConnectionStringName("AbpPermissionManagement")]
    public class BookStoreSecondDbContext : 
        AbpDbContext<BookStoreSecondDbContext>
    {
        public BookStoreSecondDbContext(
            DbContextOptions<BookStoreSecondDbContext> options)
            : base(options)
        {
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

            /* Include modules to your migration db context */
            builder.ConfigurePermissionManagement();
            builder.ConfigureSettingManagement();
            builder.ConfigureAuditLogging();
        }
    }
}
```

```

> `ConnectionStringName(...)` attribute is important here and tells to the ABP Framework which connection string should be used for this `DbContext`. We've used `AbpPermissionManagement`, but all are the same.

We need to register this `BookStoreSecondDbContext` class to the dependency injection system. Open the `BookStoreEntityFrameworkCoreModule` class in the `BookStore.EntityFrameworkCore` project and add the following line into the `ConfigureServices` method:

```
```csharp
context.Services.AddAbpDbContext<BookStoreSecondDbContext>();
````
```

We should also create a **\*\*Design Time Db Factory\*\*** class, that is used by the EF Core tooling (by `Add-Migration` and `Update-Database` PCM commands for example):

```
```csharp
using System.IO;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.Extensions.Configuration;

namespace BookStore.EntityFrameworkCore
{
    /* This class is needed for EF Core console commands
     * (like Add-Migration and Update-Database commands) */
    public class BookStoreSecondDbContextFactory
        : IDesignTimeDbContextFactory<BookStoreSecondDbContext>
    {
        public BookStoreSecondDbContext CreateDbContext(string[] args)
        {
            var configuration = BuildConfiguration();
            var builder = new DbContextOptionsBuilder<BookStoreSecondDbContext>()
                .UseSqlServer(configuration.GetConnectionString("AbpPermissionManagement"));
        }

        private static IConfigurationRoot BuildConfiguration()
        {
            var builder = new ConfigurationBuilder()
                .SetBasePath(Path.Combine(Directory.GetCurrentDirectory(),
"../BookStore.DbMigrator/"))
                .AddJsonFile("appsettings.json", optional: false);

            return builder.Build();
        }
    }
}
````
```

Now, you can open the Package Manager Console, select the `.`.EntityFrameworkCore` project as the default project (make sure the `.`.Web` project is still the startup project) and run the following command:

```
```bash
Add-Migration "Initial" -OutputDir "SecondDbMigrations" -Context BookStoreSecondDbContext
````
```

This will add a `SecondDbMigrations` folder in the `EntityFrameworkCore` project and a migration class inside it. `OutputDir` and `Context` parameters are required since we currently have two `DbContext` class and two migrations folder in the same project.

You can now run the following command to create the database and the tables inside it:

```
```bash
Update-Database -Context BookStoreSecondDbContext
```
```

A new database, named `BookStore\_SecondDb` should be created.

#### #### Remove Modules from the Main Database

We've **created a second database** that contains tables for the Audit Logging, Permission Management and Setting Management modules. So, we should **delete these tables from the main database**. It is pretty easy.

First, remove the following lines from the `BookStoreDbContext` class:

```
```csharp
builder.ConfigurePermissionManagement();
builder.ConfigureSettingManagement();
builder.ConfigureAuditLogging();
```
```

Open the Package Manager Console, select the `EntityFrameworkCore` as the Default project (make sure that the `Web` project is still the startup project) and run the following command:

```
```
Add-Migration "Removed_Audit_Setting_Permission_Modules" -Context BookStoreDbContext
```
```

This command will create a new migration class as shown below:

```
```csharp
public partial class Removed_Audit_Setting_Permission_Modules : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "AbpAuditLogActions");

        migrationBuilder.DropTable(
            name: "AbpEntityPropertyChanges");

        migrationBuilder.DropTable(
            name: "AbpPermissionGrants");

        migrationBuilder.DropTable(
            name: "AbpSettings");
    }
}
```

```

        migrationBuilder.DropTable(
            name: "AbpEntityChanges");

        migrationBuilder.DropTable(
            name: "AbpAuditLogs");
    }

    ...
}
```

```

Be careful in this step:

- \* If you have a **\*\*live system\*\***, then you should care about the **\*\*data loss\*\***. You need to move the table contents to the second database before deleting the tables.
- \* If you **\*\*haven't started\*\*** your project yet, you can consider to **\*\*remove all the migrations\*\*** and re-create the initial one to have a cleaner migration history.

Run the following command to delete the tables from your main database:

```

```bash
Update-Database -Context BookStoreDbContext
```

```

Notice that you've also **\*\*deleted some initial seed data\*\*** (for example, permission grants for the admin role) if you haven't copied it to the new database. If you run the application, you may not login anymore. The solution is simple: **\*\*Re-run the `DbMigrator` console application\*\*** in your solution, it will seed the new database.

### ### Automate the Second Database Schema Migration

``.DbMigrator`` console application can run the database seed code across multiple databases, without any additional configuration. However, it can not apply the EF Core Code First Migrations for the database of the ``BookStoreSecondDbContext``. Now, you will see how to configure the console migration application to handle both databases.

``EntityFrameworkCoreBookStoreDbSchemaMigrator`` class inside the ``Acme.BookStore.EntityFrameworkCore`` project is responsible to migrate the database schema for the ``BookStoreMigrationsDbContext``. It should be like that:

```

```csharp
using System;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
using BookStore.Data;
using Volo.Abp.DependencyInjection;

namespace BookStore.EntityFrameworkCore
{
    public class EntityFrameworkCoreBookStoreDbSchemaMigrator

```

```

    : IBookStoreDbSchemaMigrator, ITransientDependency
{
    private readonly IServiceProvider _serviceProvider;

    public EntityFrameworkCoreBookStoreDbSchemaMigrator(
        IServiceProvider serviceProvider)
    {
        _serviceProvider = serviceProvider;
    }

    public async Task MigrateAsync()
    {
        /* We intentionally resolving the BookStoreDbContext
         * from IServiceProvider (instead of directly injecting it)
         * to properly get the connection string of the current tenant in the
         * current scope.
        */

        await _serviceProvider
            .GetRequiredService<BookStoreDbContext>()
            .Database
            .MigrateAsync();
    }
}
```

```

Add the following code inside the `MigrateAsync` method:

```

```csharp
await _serviceProvider
    .GetRequiredService<BookStoreSecondDbContext>()
    .Database
    .MigrateAsync();
```

```

So, the `MigrateAsync` method should look like the following:

```

```csharp
public async Task MigrateAsync()
{
    /* We intentionally resolving the BookStoreDbContext
     * from IServiceProvider (instead of directly injecting it)
     * to properly get the connection string of the current tenant in the
     * current scope.
    */

    await _serviceProvider
        .GetRequiredService<BookStoreDbContext>()
        .Database
        .MigrateAsync();
}
```

```

```

 await _serviceProvider
 .GetRequiredService<BookStoreSecondDbContext>()
 .Database
 .MigrateAsync();
 }
````
```

That's all. You can now run the `DbMigrator` application to migrate & seed the databases. To test, you can delete both databases and run the `DbMigrator` application again to see if it creates both of the databases.

Fixing the Tests

Creating a new DbContext will break the integration tests. It is easy to fix. Open the `BookStoreEntityFrameworkCoreTestModule` class in the `BookStore.EntityFrameworkCore.Tests` project, find the `CreateDatabaseAndGetConnection` method. It should be like that:

```

```csharp
private static SqliteConnection CreateDatabaseAndGetConnection()
{
 var connection = new SqliteConnection("Data Source=:memory:");
 connection.Open();

 var options = new DbContextOptionsBuilder<BookStoreDbContext>()
 .UseSqlite(connection)
 .Options;

 using (var context = new BookStoreDbContext(options))
 {
 context.GetService<IRelationalDatabaseCreator>().CreateTables();
 }

 return connection;
}
````
```

Change it as the following:

```

```csharp
private static SqliteConnection CreateDatabaseAndGetConnection()
{
 var connection = new SqliteConnection("Data Source=:memory:");
 connection.Open();

 var options = new DbContextOptionsBuilder<BookStoreDbContext>()
 .UseSqlite(connection)
 .Options;

 using (var context = new BookStoreDbContext(options))
 {
 context.GetService<IRelationalDatabaseCreator>().CreateTables();
 }

 return connection;
}
````
```

```

    }

    // Add the following code -----
    var optionsForSecondDb = new DbContextOptionsBuilder<BookStoreSecondDbContext>()
        .UseSqlite(connection)
        .Options;

    using (var context = new BookStoreSecondDbContext(optionsForSecondDb))
    {
        context.GetService<IRelationalDatabaseCreator>().CreateTables();
    }
    //-----

    return connection;
}
```

```

Integration tests now will work. I've used the same database in the tests to keep it simple.

## ## Separating Host & Tenant Database Schemas

In a multi-tenant solution, you may want to separate your database schemas, so host-related tables don't locate in the tenant databases when tenants have separate databases.

Some pre-built ABP modules are related only with the host side, like the [Tenant Management](Modules/Tenant-Management.md) module. So, in the tenant `DbContext` class you don't call `modelBuilder.ConfigureTenantManagement()` and that's all.

Some modules, like the [Identity](Modules/Identity.md) module, is both used in host and tenant sides. It stores tenant users in the tenant database and host users in the host database. However, it stores some entities, like `IdentityClaimType`, only in the host side. In this case, you don't want to add these tables in the tenant database, even if they are not used and will always be empty for tenants.

ABP provides a simple way to set the multi-tenancy side for a `DbContext`, so the modules can check it and decide to map tables to the database, or not.

```

```csharp
public class MyTenantDbContext : AbpDbContext<MyTenantDbContext>
{
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.SetMultiTenancySide(MultiTenancySides.Tenant);

        base.OnModelCreating(modelBuilder);

        modelBuilder.ConfigureIdentity();
        modelBuilder.ConfigureFeatureManagement();
        modelBuilder.ConfigureAuditLogging();
    }
}
```

```

```
```
```

The first line in the `OnModelCreating` sets multi-tenancy side to `Tenant`. For this example, Feature management tables are not created (because all the tables are host-specific), so calling `modelBuilder.ConfigureFeatureManagement()` has no effect. Also, `ConfigureIdentity()` call respects to the multi-tenancy side and doesn't create host-specific tables for this database.

`SetMultiTenancySide` can get the following values:

- * `MultiTenancySides.Both` (**default value**): This `DbContext` (and the related database) is shared by host and tenant.
- * `MultiTenancySides.Host` : This `DbContext` (and the related database) is used only by the host side.
- * `MultiTenancySides.Tenant` : This `DbContext` (and the related database) is only for tenants.

If you create a re-usable application module or want to check that value in your application code, you can use `modelBuilder.GetMultiTenancySide()` to check the current side.

```
```csharp
```

```
var side = modelBuilder.GetMultiTenancySide();
if (!side.HasFlag(MultiTenancySides.Host))
{
 ...
}
```

```
```
```

Or practically you can use one of the shortcut extension methods:

```
```csharp
```

```
if (modelBuilder.IsTenantOnlyDatabase())
{
 ...
}
```

```
```
```

There are four methods to check the current side:

- * `IsHostDatabase()`: Returns `true` if you should create host-related tables. It is equivalent of checking
`modelBuilder.GetMultiTenancySide().HasFlag(MultiTenancySides.Host)`.
- * `IsHostOnlyDatabase()`: Returns `true` if you should only create host-related tables, but should not create tenant-related tables. It is equivalent of checking
`modelBuilder.GetMultiTenancySide() == MultiTenancySides.Host`.
- * `IsTenantDatabase()`: Returns `true` if you should create tenant-related tables. It is equivalent of checking
`modelBuilder.GetMultiTenancySide().HasFlag(MultiTenancySides.Tenant)`.
- * `IsTenantOnlyDatabase()`: Returns `true` if you should only create tenant-related tables, but should not create host-related tables. It is equivalent of checking
`modelBuilder.GetMultiTenancySide() == MultiTenancySides.Tenant`.

All pre-built ABP [modules] (Modules/Index.md) checks this value in their `modelBuilder.ConfigureXXX()` methods.

Conclusion

This document explains how to split your databases and manage your database migrations of your solution for Entity Framework Core. In brief, you need to have a separate migration project per different databases.

Source Code

You can find the source code of the example project referenced by this document [here] (<https://github.com/abpframework/abp-samples/tree/master/EfCoreMigrationDemo>). You can also find the changes explained in this document as a [single commit] (<https://github.com/abpframework/abp-samples/pull/95/commits/c2ffd76175e0a6fdfcf6477bbaea23dc2793fedd>).

11.2.2 Switch DBMS

Switch to Another DBMS for Entity Framework Core

[ABP CLI] (CLI.md) provides a `--dbms` option to allow you to choose your Database Management System (DBMS) while creating a new solution. It accepts the following values:

- `SqlServer` (default)
- `MySQL`
- `SQLite`
- `Oracle`
- `Oracle-Devart`
- `PostgreSQL`

So, if you want to use MySQL for your solution, you can use the `--dbms MySQL` option while using the `abp new` command. Example:

```
```bash
abp new BookStore --dbms MySQL
```
```

Also, the [Get Started page] (<https://abp.io/get-started>) on the ABP website allows you to select one of the providers.

> **This document provides guidance for who wants to manually change their DBMS after creating the solution.**

You can use the following documents to learn how to **switch to your favorite DBMS**:

- * [MySQL] (Entity-Framework-Core-MySQL.md)
- * [PostgreSQL] (Entity-Framework-Core-PostgreSQL.md)
- * [Oracle] (Entity-Framework-Core-Oracle.md)
- * [SQLite] (Entity-Framework-Core-SQLite.md)

You can also configure your DBMS provider ****without**** these integration packages. While using the integration package is always recommended (it also makes standard for the depended version across different modules), you can do it yourself if there is no integration package for your DBMS provider.

For an example, this document explains how to switch to MySQL without using [[the MySQL integration package](#)] (Entity-Framework-Core-MySQL.md).

Replace the SQL Server Dependency

- * Remove the [\[Volo.Abp.EntityFrameworkCore.SqlServer\]](#) (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer>) NuGet package dependency from the `.`.EntityFrameworkCore` project.
- * Add the [\[Pomelo.EntityFrameworkCore.MySql\]](#) (<https://www.nuget.org/packages/Pomelo.EntityFrameworkCore.MySql/>) NuGet package dependency to your `.`.EntityFrameworkCore` project.

Remove the Module Dependency

Remove the `AbpEntityFrameworkCoreSqlServerModule` from the dependency list of your ****YourProjectName**EntityFrameworkCoreModule**** class.

Change the UseSqlServer() Calls

Find the following code part inside the ***YourProjectName*EntityFrameworkCoreModule** class:

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
 options.UseSqlServer();
});
```

Replace it with the following code part:

```
```csharp
Configure<AbpDbContextOptions>(options =>
{
    options.Configure(ctx =>
    {
        if (ctx.ExistingConnection != null)
        {
            ctx.DbContextOptions.UseMySql(ctx.ExistingConnection);
        }
        else
        {
            ctx.DbContextOptions.UseMySql(ctx.ConnectionString);
        }
    });
});
```

* `UseMySql` calls in this code is defined by the Pomelo.EntityFrameworkCore.MySql package and you can use its additional options if you need.

* This code first checks if there is an existing (active) connection to the same database in the current request and reuses it if possible. This allows to share a single transaction among different DbContext types. ABP handles the rest of the things.

* It uses `ctx.ConnectionString` and passes to the `UseMySql` if there is no active connection (which will cause to create a new database connection). Using the `ctx.ConnectionString` is important here. Don't pass a static connection string (or a connection string from a configuration). Because ABP [dynamically determines the correct connection string](Connection-Strings.md) in a multi-database or [multi-tenant](Multi-Tenancy.md) environment.

Change the Connection Strings

MySQL connection strings are different than SQL Server connection strings. So, check all `appsettings.json` files in your solution and replace the connection strings inside them. See the [connectionstrings.com](https://www.connectionstrings.com/mysql/) for details of MySQL connection string options.

You typically will change the `appsettings.json` inside the `DbMigrator` and `Web` projects, but it depends on your solution structure.

DBMS restrictions

MySQL DBMS has some slight differences than the SQL Server. Some module database mapping configuration (especially the field lengths) causes problems with MySQL. For example, some of the the [IdentityServer module](Modules/IdentityServer.md) tables has such problems and it provides an option to configure the fields based on your DBMS.

The module may provide some built-in solutions. You can configure it via `ModelBuilder`. eg: `Auth Server` module.

```
```csharp
builder.ConfigureIdentityServer(options =>
{
 options.DatabaseProvider = EfCoreDatabaseProvider.MySql;
});```

```

Then `ConfigureIdentityServer()` method will set the field lengths to not exceed the MySQL limits. Refer to related module documentation if you have any problem while creating or executing the database migrations.

## ## Re-Generate the Migrations

The startup template uses [Entity Framework Core's Code First Migrations](https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/). EF Core Migrations depend on the selected DBMS provider. So, changing the DBMS provider will cause the migration fails.

- \* Delete the Migrations folder under the ` `.EntityFrameworkCore` project and re-build the solution.
- \* Run `Add-Migration "Initial"` on the Package Manager Console (select the ` `.DbMigrator` (or ` `.Web` ) project as the startup project in the Solution Explorer and select the ` `.EntityFrameworkCore` project as the default project in the Package Manager Console).

This will create a database migration with all database objects (tables) configured.

Run the ` `.DbMigrator` project to create the database and seed the initial data.

#### ## Run the Application

It is ready. Just run the application and enjoy coding.

Related discussions: <https://github.com/abpframework/abp/issues/1920>

#### 11.2.2.1 To MySQL

##### # Switch to EF Core MySQL Provider

> [ABP CLI] (CLI.md) and the [Get Started] (<https://abp.io/get-started>) page already provides an option to create a new solution with MySQL. See [that document] (Entity-Framework-Core-Other-DBMS.md) to learn how to use. This document provides guidance for who wants to manually switch to MySQL after creating the solution.

This document explains how to switch to the \*\*MySQL\*\* database provider for \*\*[the application startup template]\*\* ([Startup-Templates/Application.md](#)) which comes with SQL Server provider pre-configured.

##### ## Replace the Volo.Abp.EntityFrameworkCore.SqlServer Package

` `.EntityFrameworkCore` project in the solution depends on the [Volo.Abp.EntityFrameworkCore.SqlServer] (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer>) NuGet package. Remove this package and add the same version of the [Volo.Abp.EntityFrameworkCore.MySQL] (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.MySQL>) package.

##### ## Replace the Module Dependency

Find \*\*\*YourProjectName\*EntityFrameworkCoreModule\*\*\* class inside the ` `.EntityFrameworkCore` project, remove ` `typeof(AbpEntityFrameworkCoreSqlServerModule)` ` from the ` `DependsOn` ` attribute, add ` `typeof(AbpEntityFrameworkCoreMySQLModule)` ` (also replace ` `using Volo.Abp.EntityFrameworkCore.SqlServer;` ` with ` `using Volo.Abp.EntityFrameworkCore.MySQL;` `).

##### ## UseMySQL()

Find ` `UseSqlServer()` ` calls in your solution. Check the following files:

\* \*YourProjectName\*EntityFrameworkCoreModule.cs inside the ` `.EntityFrameworkCore` project.  
Replace ` `UseSqlServer()` ` with ` `UseMySQL()` `.

\* \*YourProjectName\*DbContextFactory.cs inside the `\*.EntityFrameworkCore` project. Replace `UseSqlServer()` with `UseMySql()`. Then add a new parameter (`ServerVersion`) to `UseMySql()` method. Example: `.`.UseMySql(configuration.GetConnectionString("Default"), ServerVersion.Parse("8.0.21-mysql"))`. See [this issue](<https://github.com/PomeloFoundation/Pomelo.EntityFrameworkCore.MySql/pull/1233>) for more information about `ServerVersion`)

> Depending on your solution structure, you may find more code files need to be changed.

## ## Change the Connection Strings

MySQL connection strings are different than SQL Server connection strings. So, check all `appsettings.json` files in your solution and replace the connection strings inside them. See the [connectionstrings.com](<https://www.connectionstrings.com/mysql/>) for details of MySQL connection string options.

You typically will change the `appsettings.json` inside the `\*.DbMigrator` and `\*.Web` projects, but it depends on your solution structure.

## ## Re-Generate the Migrations

The startup template uses [Entity Framework Core's Code First Migrations](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). EF Core Migrations depend on the selected DBMS provider. So, changing the DBMS provider will cause the migration fails.

- \* Delete the Migrations folder under the `\*.EntityFrameworkCore` project and re-build the solution.
- \* Run `Add-Migration "Initial"` on the Package Manager Console (select the `\*.DbMigrator` (or `\*.Web`) project as the startup project in the Solution Explorer and select the `\*.EntityFrameworkCore` project as the default project in the Package Manager Console).

This will create a database migration with all database objects (tables) configured.

Run the `\*.DbMigrator` project to create the database and seed the initial data.

## ## Run the Application

It is ready. Just run the application and enjoy coding.

### 11.2.2.2 To PostgreSQL

#### # Switch to EF Core PostgreSQL Provider

> [ABP CLI](CLI.md) and the [Get Started](<https://abp.io/get-started>) page already provides an option to create a new solution with PostgreSQL. See [that document](Entity-Framework-Core-Other-DBMS.md) to learn how to use. This document provides guidance for who wants to manually switch to PostgreSQL after creating the solution.

This document explains how to switch to the **PostgreSQL** database provider for **[the application startup template](Startup-Templates/Application.md)** which comes with SQL Server provider pre-configured.

## ## Replace the Volo.Abp.EntityFrameworkCore.SqlServer Package

`.EntityFrameworkCore` project in the solution depends on the [\[Volo.Abp.EntityFrameworkCore.SqlServer\]](https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer) (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer>) NuGet package. Remove this package and add the same version of the [\[Volo.Abp.EntityFrameworkCore.PostgreSql\]](https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.PostgreSql) (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.PostgreSql>) package.

## ## Replace the Module Dependency

Find **\*YourProjectName\*EntityFrameworkCoreModule** class inside the `.EntityFrameworkCore` project, remove `typeof(AbpEntityFrameworkCoreSqlServerModule)` from the `DependsOn` attribute, add `typeof(AbpEntityFrameworkCorePostgreSqlModule)` (also replace `using Volo.Abp.EntityFrameworkCore.SqlServer;` with `using Volo.Abp.EntityFrameworkCore.PostgreSql;`).

## ## UseNpgsql()

Find `UseSqlServer()` call in **\*YourProjectName\*EntityFrameworkCoreModule.cs** inside the `.EntityFrameworkCore` project and replace with `UseNpgsql()`.

Find `UseSqlServer()` call in **\*YourProjectName\*DbContextFactory.cs** inside the `.EntityFrameworkCore` project and replace with `UseNpgsql()`.

> Depending on your solution structure, you may find more `UseSqlServer()` calls that needs to be changed.

## ## Change the Connection Strings

PostgreSQL connection strings are different than SQL Server connection strings. So, check all `appsettings.json` files in your solution and replace the connection strings inside them. See the [\[connectionstrings.com\]](https://www.connectionstrings.com/postgresql/) (<https://www.connectionstrings.com/postgresql/>) for details of PostgreSQL connection string options.

You typically will change the `appsettings.json` inside the `DbMigrator` and `Web` projects, but it depends on your solution structure.

## ## Re-Generate the Migrations

The startup template uses [\[Entity Framework Core's Code First Migrations\]](https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/) (<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). EF Core Migrations depend on the selected DBMS provider. So, changing the DBMS provider will cause the migration fails.

\* Delete the Migrations folder under the `EntityFrameworkCore` project and re-build the solution.

\* Run `Add-Migration "Initial"` on the Package Manager Console (select the `DbMigrator` (or `Web`) project as the startup project in the Solution Explorer and select the `EntityFrameworkCore` project as the default project in the Package Manager Console).

This will create a database migration with all database objects (tables) configured.

Run the `DbMigrator` project to create the database and seed the initial data.

#### ## Run the Application

It is ready. Just run the application and enjoy coding.

### 11.2.2.3 To Oracle

#### # Switch to EF Core Oracle Provider

> [ABP CLI](CLI.md) and the [Get Started](https://abp.io/get-started) page already provides an option to create a new solution with Oracle. See [that document](Entity-Framework-Core-Other-DBMS.md) to learn how to use. This document provides guidance for who wants to manually switch to Oracle after creating the solution.

This document explains how to switch to the \*\*Oracle\*\* database provider for \*\*[the application startup template](Startup-Templates/Application.md)\*\* which comes with SQL Server provider pre-configured.

ABP Framework provides integrations for two different Oracle packages. See one of the following documents based on your provider decision:

- \* \*\*[`Volo.Abp.EntityFrameworkCore.Oracle`](Entity-Framework-Core-Oracle-Official.md)\*\* package uses the official & free oracle driver.
- \* \*\*[`Volo.Abp.EntityFrameworkCore.Oracle.Devart`](Entity-Framework-Core-Oracle-Devart.md)\*\* package uses the commercial (paid) driver of [Devart](https://www.devart.com/) company.

> You can choose one of the package you want. If you don't know the differences of the packages, please search for it. ABP Framework only provides integrations it doesn't provide support for such 3rd-party libraries.

#### 11.2.2.3.1 Oracle (Official)

#### # Switch to EF Core Oracle Provider

This document explains how to switch to the \*\*Oracle\*\* database provider for \*\*[the application startup template](Startup-Templates/Application.md)\*\* which comes with SQL Server provider pre-configured.

> Before switching your provider, please ensure your Oracle version is \*\*v12.2\*\*. In the earlier versions of Oracle, there were long identifier limitations that prevents creating

a database table, column or index longer than 30 bytes. With [v12.2] (<https://docs.oracle.com/en/database/oracle/oracle-database/12.2/newft/features.html#GUID-64283AD6-0939-47B0-856E-5E9255D7246B>) "The maximum length of identifiers is increased to 128 bytes". \*\*v12.2\*\* and later versions, you can use the database tables, columns and indexes provided by ABP without any problems.

## Replace the Volo.Abp.EntityFrameworkCore.SqlServer Package

`.EntityFrameworkCore` project in the solution depends on the [Volo.Abp.EntityFrameworkCore.SqlServer] (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer>) NuGet package. Remove this package and add the same version of the [Volo.Abp.EntityFrameworkCore.Oracle] (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.Oracle>) package.

## Replace the Module Dependency

Find **\*YourProjectName\*EntityFrameworkModule** class inside the `.EntityFrameworkCore` project, remove `typeof(AbpEntityFrameworkCoreSqlServerModule)` from the `DependsOn` attribute, add `typeof(AbpEntityFrameworkCoreOracleModule)`

Also replace `using Volo.Abp.EntityFrameworkCore.SqlServer;` with `using Volo.Abp.EntityFrameworkCore.Oracle;` .

## UseOracle()

Find `UseSqlServer()` calls in your solution, replace with `UseOracle()`. Check the following files:

- \* **\*YourProjectName\*EntityFrameworkModule.cs** inside the `.EntityFrameworkCore` project.
- \* **\*YourProjectName\*DbContextFactory.cs** inside the `.EntityFrameworkCore` project.

In the `CreateDbContext()` method of the **\*YourProjectName\*DbContextFactory.cs**, replace the following code block

```
```csharp
var builder = new DbContextOptionsBuilder<YourProjectNameDbContext>()
    .UseSqlServer(configuration.GetConnectionString("Default"));
```
```

with this one (just changes `UseSqlServer(...)` to `UseOracle(...)`)

```
```csharp
var builder = new DbContextOptionsBuilder<YourProjectNameDbContext>()
    .UseOracle(configuration.GetConnectionString("Default"));
```
```

➤ Depending on your solution structure, you may find more code files need to be changed.

## Change the Connection Strings

Oracle connection strings are different than SQL Server connection strings. So, check all `appsettings.json` files in your solution and replace the connection strings inside them.

See the [connectionstrings.com](<https://www.connectionstrings.com/oracle/>) for details of Oracle connection string options.

You typically will change the `appsettings.json` inside the `DbMigrator` and `Web` projects, but it depends on your solution structure.

## ## Re-Generate the Migrations

The startup template uses [Entity Framework Core's Code First Migrations](<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>) by default.

EF Core Migrations depend on the selected DBMS provider. Changing the DBMS provider, may not work with the existing migrations.

- \* Delete the `Migrations` folder under the `EntityFrameworkCore` project and re-build the solution.
- \* Run `Add-Migration "Initial"` on the Package Manager Console window (select the `DbMigrator` (or `Web`) project as the startup project in the Solution Explorer and select the `EntityFrameworkCore` project as the default project in the Package Manager Console).

This will scaffold a new migration for Oracle.

Run the `DbMigrator` project to create the database, apply the changes and seed the initial data.

## ## Run the Application

It is ready. Just run the application and enjoy coding.

### 11.2.2.3.2 Oracle (Devart)

#### # Switch to EF Core Oracle Devart Provider

This document explains how to switch to the \*\*Oracle\*\* database provider for \*\*[the application startup template](Startup-Templates/Application.md)\*\* which comes with SQL Server provider pre-configured.

- > This document uses a paid library of [Devart](<https://www.devart.com/dotconnect/oracle/>) company, See [this document](Entity-Framework-Core-Oracle.md) for other options.
- > Before switching your provider, please ensure your Oracle version is \*\*v12.2\*\*. In the earlier versions of Oracle, there were long identifier limitations that prevents creating a database table, column or index longer than 30 bytes. With [v12.2](<https://docs.oracle.com/en/database/oracle/oracle-database/12.2/newft/features.html#GUID-64283AD6-0939-47B0-856E-5E9255D7246B>) "The maximum length of identifiers is increased to 128 bytes". \*\*v12.2\*\* and later versions, you can use the database tables, columns and indexes provided by ABP without any problems.

## ## Replace the Volo.Abp.EntityFrameworkCore.SqlServer Package

``.EntityFrameworkCore`` project in the solution depends on the `[Volo.Abp.EntityFrameworkCore.SqlServer]` (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer>) NuGet package. Remove this package and add the same version of the `[Volo.Abp.EntityFrameworkCore.Oracle.Devart]` (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.Oracle.Devart>) package.

#### ## Replace the Module Dependency

Find `***YourProjectName*EntityFrameworkCoreModule***` class inside the ``.EntityFrameworkCore`` project, remove ``typeof(AbpEntityFrameworkCoreSqlServerModule)`` from the ``DependsOn`` attribute, add ``typeof(AbpEntityFrameworkCoreOracleDevartModule)``

Also replace ``using Volo.Abp.EntityFrameworkCore.SqlServer;`` with ``using Volo.Abp.EntityFrameworkCore.Oracle.Devart;``.

#### ## UseOracle()

Find ``UseSqlServer()`` calls in your solution, replace with ``UseOracle()``. Check the following files:

- \* `*YourProjectName*EntityFrameworkCoreModule.cs` inside the ``.EntityFrameworkCore`` project.
- \* `*YourProjectName*DbContextFactory.cs` inside the ``.EntityFrameworkCore`` project.

In the ``CreateDbContext()`` method of the `*YourProjectName*DbContextFactory.cs`, replace the following code block

```
```csharp
var builder = new DbContextOptionsBuilder<YourProjectNameDbContext>()
    .UseSqlServer(configuration.GetConnectionString("Default"));
```
```

with this one

```
```csharp
var builder = (DbContextOptionsBuilder<YourProjectNameDbContext>)
    new DbContextOptionsBuilder<YourProjectNameDbContext>().UseOracle
    (
        configuration.GetConnectionString("Default")
    );
```
```

➤ Depending on your solution structure, you may find more code files need to be changed.

#### ## Change the Connection Strings

Oracle connection strings are different than SQL Server connection strings. So, check all ``appsettings.json`` files in your solution and replace the connection strings inside them. See the [\[connectionstrings.com\]](https://www.connectionstrings.com/oracle/) (<https://www.connectionstrings.com/oracle/>) for details of Oracle connection string options.

You typically will change the `appsettings.json` inside the `DbMigrator` and `Web` projects, but it depends on your solution structure.

## ## Re-Generate the Migrations

The startup template uses [Entity Framework Core's Code First Migrations] (<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>) by default.

EF Core Migrations depend on the selected DBMS provider. Changing the DBMS provider, may not work with the existing migrations.

- \* Delete the `Migrations` folder under the `EntityFrameworkCore` project and re-build the solution.
- \* Run `Add-Migration "Initial"` on the Package Manager Console window (select the `DbMigrator` (or `Web`) project as the startup project in the Solution Explorer and select the `EntityFrameworkCore` project as the default project in the Package Manager Console).

This will scaffold a new migration for Oracle.

Run the `DbMigrator` project to create the database, apply the changes and seed the initial data.

## ## Run the Application

It is ready. Just run the application and enjoy coding.

### 11.2.2.4 To SQLite

#### # Switch to EF Core SQLite Provider

> [ABP CLI] (CLI.md) and the [Get Started] (<https://abp.io/get-started>) page already provides an option to create a new solution with SQLite. See [[that document](#)] (Entity-Framework-Core-Other-DBMS.md) to learn how to use. This document provides guidance for who wants to manually switch to SQLite after creating the solution.

This document explains how to switch to the \*\*SQLite\*\* database provider for \*\*[the application startup template]\*\* ([Startup-Templates/Application.md](#)) which comes with SQL Server provider pre-configured.

## ## Replace the Volo.Abp.EntityFrameworkCore.SqlServer Package

`EntityFrameworkCore` project in the solution depends on the [Volo.Abp.EntityFrameworkCore.SqlServer] (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.SqlServer>) NuGet package. Remove this package and add the same version of the [Volo.Abp.EntityFrameworkCore.Sqlite] (<https://www.nuget.org/packages/Volo.Abp.EntityFrameworkCore.Sqlite>) package.

## ## Replace the Module Dependency

Find **`***YourProjectName*EntityFrameworkCoreModule**`** class inside the ``.EntityFrameworkCore`` project, remove ``typeof(AbpEntityFrameworkCoreSqlServerModule)`` from the ``DependsOn`` attribute, add ``typeof(AbpEntityFrameworkCoreSqliteModule)`` (also replace ``using Volo.Abp.EntityFrameworkCore.SqlServer;`` with ``using Volo.Abp.EntityFrameworkCore.Sqlite;``).

## ## UseSqlite()

Find ``UseSqlServer()`` calls in your solution, replace with ``UseSqlite()``. Check the following files:

- \* `*YourProjectName*EntityFrameworkCoreModule.cs` inside the ``.EntityFrameworkCore`` project.
- \* `*YourProjectName*DbContextFactory.cs` inside the ``.EntityFrameworkCore`` project.

> Depending on your solution structure, you may find more code files need to be changed.

## ## Change the Connection Strings

SQLite connection strings are different than SQL Server connection strings. So, check all ``appsettings.json`` files in your solution and replace the connection strings inside them. See the [\[connectionstrings.com\]](#) (<https://www.connectionstrings.com/sqlite/>) for details of SQLite connection string options.

An example connection string is

```
...
{
 "ConnectionStrings": {
 "Default": "Filename=./MySQLiteDBFile.sqlite"
 }
}...
```

You typically will change the ``appsettings.json`` inside the ``.DbMigrator`` and ``.Web`` projects, but it depends on your solution structure.

## ## Re-Generate the Migrations

The startup template uses [\[Entity Framework Core's Code First Migrations\]](#) (<https://docs.microsoft.com/en-us/ef/core/managing-schemas/migrations/>). EF Core Migrations depend on the selected DBMS provider. So, changing the DBMS provider will cause the migration fails.

- \* Delete the Migrations folder under the ``.EntityFrameworkCore`` project and re-build the solution.
- \* Run ``Add-Migration "Initial"`` on the Package Manager Console (select the ``.DbMigrator`` (or ``.Web``) project as the startup project in the Solution Explorer and select the ``.EntityFrameworkCore`` project as the default project in the Package Manager Console).

This will create a database migration with all database objects (tables) configured.

Run the ``.DbMigrator`` project to create the database and seed the initial data.

## ## Run the Application

It is ready. Just run the application and enjoy coding.

## 11.3 MongoDB

### # MongoDB Integration

This document explains how to integrate MongoDB as a database provider to ABP based applications and how to configure it.

#### ## Installation

`[Volo.Abp.MongoDB](#)` is the main NuGet package for the MongoDB integration. Install it to your project (for a layered application, to your data/infrastructure layer), You can use the [\[ABP CLI\]](#)(CLI.md) to install it to your project. Execute the following command in the folder of the .csproj file of the layer:

```
```  
abp add-package Volo.Abp.MongoDB  
```
```

> If you haven't done it yet, you first need to install the [\[ABP CLI\]](#)(CLI.md). For other installation options, see [\[the package description page\]](#)(<https://abp.io/package-detail/Volo.Abp.MongoDB>).

Then add `AbpMongoDbModule` module dependency to your [\[module\]](#)(Module-Development-Basics.md) :

```
```c#  
using Volo.Abp.MongoDB;  
using Volo.Abp.Modularity;  
  
namespace MyCompany.MyProject  
{  
    [DependsOn(typeof(AbpMongoDbModule))]  
    public class MyModule : AbpModule  
    {  
        //...  
    }  
}```
```

Creating a Mongo Db Context

ABP introduces ****Mongo Db Context**** concept (which is similar to Entity Framework Core's DbContext) to make it easier to use collections and configure them. An example is shown below:

```

```c#
public class MyDbContext : AbpMongoDbContext
{
 public IMongoCollection<Question> Questions => Collection<Question>();

 public IMongoCollection<Category> Categories => Collection<Category>();

 protected override void CreateModel(IMongoModelBuilder modelBuilder)
 {
 base.CreateModel(modelBuilder);

 //Customize the configuration for your collections.
 }
}
```
* It's derived from `AbpMongoDbContext` class.
* Adds a public `IMongoCollection<TEntity>` property for each mongo collection. ABP uses these properties to create default repositories by default.
* Overriding `CreateModel` method allows to configure collection configuration.

```

Configure Mapping for a Collection

ABP automatically register entities to MongoDB client library for all `IMongoCollection<TEntity>` properties in your DbContext. For the example above, `Question` and `Category` entities are automatically registered.

For each registered entity, it calls `AutoMap()` and configures known properties of your entity. For instance, if your entity implements `IHasExtraProperties` interface (which is already implemented for every aggregate root by default), it automatically configures `ExtraProperties`.

So, most of times you don't need to explicitly configure registration for your entities. However, if you need it you can do it by overriding the `CreateModel` method in your DbContext. Example:

```

```csharp
protected override void CreateModel(IMongoModelBuilder modelBuilder)
{
 base.CreateModel(modelBuilder);

 modelBuilder.Entity<Question>(b =>
 {
 b.CollectionName = "MyQuestions"; //Sets the collection name
 b.BsonMap.UnmapProperty(x => x.MyProperty); //Ignores 'MyProperty'
 });
}
```

```

This example changes the mapped collection name to 'MyQuestions' in the database and ignores a property in the `Question` class.

If you only need to configure the collection name, you can also use ` `[MongoCollection]` attribute for the collection in your DbContext. Example:

```
```csharp
[MongoCollection("MyQuestions")] //Sets the collection name
public IMongoCollection<Question> Questions => Collection<Question>();
````
```

Configure the Connection String Selection

If you have multiple databases in your application, you can configure the connection string name for your DbContext using the ` `[ConnectionStringName]` attribute. Example:

```
```csharp
[ConnectionStringName("MySecondConnString")]
public class MyDbContext : AbpMongoDbContext
{
}
````
```

If you don't configure, the ` `Default` ` connection string is used. If you configure a specific connection string name, but not define this connection string name in the application configuration then it fallbacks to the ` `Default` ` connection string.

Registering DbContext To Dependency Injection

Use ` `AddAbpDbContext` ` method in your module to register your DbContext class for [dependency injection] (Dependency-Injection.md) system.

```
```c#
using Microsoft.Extensions.DependencyInjection;
using Volo.Abp.MongoDB;
using Volo.Abp.Modularity;

namespace MyCompany.MyProject
{
 [DependsOn(typeof(AbpMongoDbContext))]
 public class MyModule : AbpModule
 {
 public override void ConfigureServices(ServiceConfigurationContext context)
 {
 context.Services.AddMongoDbContext<MyDbContext>();

 //...
 }
 }
}
````
```

Add Default Repositories

ABP can automatically create default [generic repositories] (Repositories.md) for the entities in your DbContext. Just use `AddDefaultRepositories()` option on the registration:

```
```c#
services.AddMongoDbContext<MyDbContext>(options =>
{
 options.AddDefaultRepositories();
});
```

This will create a repository for each [aggregate root entity] (Entities.md) (classes derived from `AggregateRoot`) by default. If you want to create repositories for other entities too, then set `includeAllEntities` to `true`:

```
```c#
services.AddMongoDbContext<MyDbContext>(options =>
{
    options.AddDefaultRepositories(includeAllEntities: true);
});
```

Then you can inject and use ` IRepository< TEntity, TPrimaryKey >` in your services. Assume that you have a `Book` entity with `Guid` primary key:

```
```csharp
public class Book : AggregateRoot<Guid>
{
 public string Name { get; set; }

 public BookType Type { get; set; }
}
```

(`BookType` is a simple `enum` here) And you want to create a new `Book` entity in a [domain service] (Domain-Services.md) :

```
```csharp
public class BookManager : DomainService
{
    private readonly IRepository<Book, Guid> _bookRepository;

    public BookManager(IRepository<Book, Guid> bookRepository) //inject default repository
    {
        _bookRepository = bookRepository;
    }

    public async Task<Book> CreateBook(string name, BookType type)
    {
        Check.NotNullOrWhiteSpace(name, nameof(name));

        var book = new Book
```

```

    {
        Id = GuidGenerator.Create(),
        Name = name,
        Type = type
    };

    await _bookRepository.InsertAsync(book); //Use a standard repository method

    return book;
}
```

```

This sample uses `InsertAsync` method to insert a new entity to the database.

#### ### Add Custom Repositories

Default generic repositories are powerful enough in most cases (since they implement `IQueryable`). However, you may need to create a custom repository to add your own repository methods.

Assume that you want to delete all books by type. It's suggested to define an interface for your custom repository:

```

```csharp
public interface IBookRepository : IRepository<Book, Guid>
{
    Task DeleteBooksByType(
        BookType type,
        CancellationToken cancellationToken = default(CancellationToken)
    );
}
```

```

You generally want to derive from the ` IRepository` to inherit standard repository methods. However, you don't have to. Repository interfaces are defined in the domain layer of a layered application. They are implemented in the data/infrastructure layer (`MongoDB` project in a [startup template](<https://abp.io/Templates>)).

Example implementation of the ` IBookRepository` interface:

```

```csharp
public class BookRepository :
    MongoDbRepository<BookStoreMongoDbContext, Book, Guid>,
    IBookRepository
{
    public BookRepository(IMongoDbContextProvider<BookStoreMongoDbContext>
        dbContextProvider)
        : base(dbContextProvider)
    {
    }
}
```

```

```

public async Task DeleteBooksByType(
 BookType type,
 CancellationToken cancellationToken = default(CancellationToken))
{
 var collection = await GetCollectionAsync(cancellationToken);
 await collection.DeleteManyAsync(
 Builders<Book>.Filter.Eq(b => b.Type, type),
 cancellationToken
);
}
```

```

Now, it's possible to [inject](Dependency-Injection.md) the `IBookRepository` and use the `DeleteBooksByType` method when needed.

Override Default Generic Repository

Even if you create a custom repository, you can still inject the default generic repository (` IRepository<Book, Guid>` for this example). Default repository implementation will not use the class you have created.

If you want to replace default repository implementation with your custom repository, do it inside `AddMongoDbContext` options:

```

```csharp
context.Services.AddMongoDbContext<BookStoreMongoDbContext>(options =>
{
 options.AddDefaultRepositories();
 options.AddRepository<Book, BookRepository>(); //Replaces IRepository<Book, Guid>
});
```

```

This is especially important when you want to **override a base repository method** to customize it. For instance, you may want to override `DeleteAsync` method to delete an entity in a more efficient way:

```

```csharp
public async override Task DeleteAsync(
 Guid id,
 bool autoSave = false,
 CancellationToken cancellationToken = default)
{
 //TODO: Custom implementation of the delete method
}
```

```

Access to the MongoDB API

In most cases, you want to hide MongoDB APIs behind a repository (this is the main purpose of the repository). However, if you want to access the MongoDB API over the repository,

you can use `GetDatabaseAsync()`, `GetCollectionAsync()` or `GetAggregateAsync()` extension methods. Example:

```
```csharp
public class BookService
{
 private readonly IRepository<Book, Guid> _bookRepository;

 public BookService(IRepository<Book, Guid> bookRepository)
 {
 _bookRepository = bookRepository;
 }

 public async Task FooAsync()
 {
 IMongoDatabase database = await _bookRepository.GetDatabaseAsync();
 IMongoCollection<Book> books = await _bookRepository.GetCollectionAsync();
 IAggregateFluent<Book> bookAggregate = await _bookRepository.GetAggregateAsync();
 }
}
```

```

➤ Important: You must reference to the `Volo.Abp.MongoDB` package from the project you want to access to the MongoDB API. This breaks encapsulation, but this is what you want in that case.

Transactions

MongoDB supports multi-document transactions starting from the version 4.0 and the ABP Framework supports it. However, the [startup template](Startup-templates/Index.md) **disables** transactions by default. If your MongoDB **server** supports transactions, you can enable the it in the *YourProjectMongoDbModule* class:

```
```csharp
Configure<AbpUnitOfWorkDefaultOptions>(options =>
{
 options.TransactionBehavior = UnitOfWorkTransactionBehavior.Auto;
});
```

```

➤ Or you can delete this code since this is already the default behavior.

Advanced Topics

Controlling the Multi-Tenancy

If your solution is [multi-tenant](Multi-Tenancy.md), tenants may have **separate databases**, you have **multiple** `DbContext` classes in your solution and some of your `DbContext` classes should be usable **only from the host side**, it is suggested to add `[IgnoreMultiTenancy]` attribute on your `DbContext` class. In this case, ABP guarantees that the related `DbContext` always uses the host [connection string](Connection-Strings.md), even if you are in a tenant context.

Example:

```
```csharp
[IgnoreMultiTenancy]
public class MyDbContext : AbpMongoDbContext
{
 ...
}
```

Do not use the ` [IgnoreMultiTenancy]` attribute if any one of your entities in your `DbContext` can be persisted in a tenant database.

> When you use repositories, ABP already uses the host database for the entities don't implement the `IMultiTenant` interface. So, most of time you don't need to ` [IgnoreMultiTenancy]` attribute if you are using the repositories to work with the database.

## #### Set Default Repository Classes

Default generic repositories are implemented by `MongoDbRepository` class by default. You can create your own implementation and use it for default repository implementation.

First, define your repository classes like that:

```
```csharp
public class MyRepositoryBase<TEntity>
    : MongoDbRepository<BookStoreMongoDbContext, TEntity>
    where TEntity : class, IEntity
{
    public MyRepositoryBase(IMongoDbContextProvider<BookStoreMongoDbContext>
dbContextProvider)
        : base(dbContextProvider)
    {
    }
}

public class MyRepositoryBase<TEntity, TKey>
    : MongoDbRepository<BookStoreMongoDbContext, TEntity, TKey>
    where TEntity : class, IEntity<TKey>
{
    public MyRepositoryBase(IMongoDbContextProvider<BookStoreMongoDbContext>
dbContextProvider)
        : base(dbContextProvider)
    {
    }
}
```

First one is for [entities with composite keys](Entities.md), second one is for entities with single primary key.

It's suggested to inherit from the `MongoDBRepository` class and override methods if needed. Otherwise, you will have to implement all standard repository methods manually.

Now, you can use `SetDefaultRepositoryClasses` option:

```
```csharp
context.Services.AddMongoDbContext<BookStoreMongoDbContext>(options =>
{
 options.SetDefaultRepositoryClasses(
 typeof(MyRepositoryBase<, >),
 typeof(MyRepositoryBase<>)
);
 //...
});
```

#### #### Set Base MongoDBContext Class or Interface for Default Repositories

If your MongoDBContext inherits from another MongoDBContext or implements an interface, you can use that base class or interface as the MongoDBContext for default repositories. Example:

```
```csharp
public interface IBookStoreMongoDbContext : IAbpMongoDbContext
{
    Collection<Book> Books { get; }
}
```
```

`IBookStoreMongoDbContext` is implemented by the `BookStoreMongoDbContext` class. Then you can use generic overload of the `AddDefaultRepositories` :

```
```csharp
context.Services.AddMongoDbContext<BookStoreMongoDbContext>(options =>
{
    options.AddDefaultRepositories<IBookStoreMongoDbContext>();
    //...
});
```

Now, your custom `BookRepository` can also use the `IBookStoreMongoDbContext` interface:

```
```csharp
public class BookRepository
 : MongoDBRepository<IBookStoreMongoDbContext, Book, Guid>,
 IBookRepository
{
 //...
}
```

One advantage of using interface for a `MongoDbContext` is then it becomes replaceable by another implementation.

#### #### Replace Other DbContextes

Once you properly define and use an interface for a `MongoDbContext` , then any other implementation can use the following ways to replace it:

#### #### ReplaceDbContext Attribute

```
```csharp
[ReplaceDbContext(typeof(IBookStoreMongoDbContext))]
public class OtherMongoDbContext : AbpMongoDbContext, IBookStoreMongoDbContext
{
    //...
}
```

```

#### #### ReplaceDbContext Option

```
```csharp
context.Services.AddMongoDbContext<OtherMongoDbContext>(options =>
{
    //...
    options.ReplaceDbContext<IBookStoreMongoDbContext>();
});
```

```

In this example, ``OtherMongoDbContext`` implements ``IBookStoreMongoDbContext``. This feature allows you to have multiple `MongoDbContext` (one per module) on development, but single `MongoDbContext` (implements all interfaces of all `MongoDbContexts`) on runtime.

#### #### Replacing with Multi-Tenancy

It is also possible to replace a `DbContext` based on the [multi-tenancy](Multi-Tenancy.md) side. ``ReplaceDbContext`` attribute and ``ReplaceDbContext`` method can get a ``MultiTenancySides`` option with a default value of ``MultiTenancySides.Both``.

**\*\*Example:\*\*** Replace `DbContext` only for tenants, using the ``ReplaceDbContext`` attribute

```
```csharp
[ReplaceDbContext(typeof(IBookStoreDbContext), MultiTenancySides.Tenant)]
```

```

**\*\*Example:\*\*** Replace `DbContext` only for the host side, using the ``ReplaceDbContext`` method

```
```csharp
options.ReplaceDbContext<IBookStoreDbContext>(MultiTenancySides.Host);
```

```

### ## Customize Bulk Operations

If you have better logic or using an external library for bulk operations, you can override the logic via implementing `IMongoDbBulkOperationProvider`.

- You may use example template below:

```
```csharp
public class MyCustomMongoDbBulkOperationProvider
    : IMongoDbBulkOperationProvider, ITransientDependency
{
    public async Task DeleteManyAsync<TEntity>(
        IMongoDbRepository<TEntity> repository,
        IEnumerable<TEntity> entities,
        IClientSessionHandle sessionHandle,
        bool autoSave,
        CancellationToken cancellationToken)
        where TEntity : class, IEntity
    {
        // Your logic here.
    }

    public async Task InsertManyAsync<TEntity>(
        IMongoDbRepository<TEntity> repository,
        IEnumerable<TEntity> entities,
        IClientSessionHandle sessionHandle,
        bool autoSave,
        CancellationToken cancellationToken)
        where TEntity : class, IEntity
    {
        // Your logic here.
    }

    public async Task UpdateManyAsync<TEntity>(
        IMongoDbRepository<TEntity> repository,
        IEnumerable<TEntity> entities,
        IClientSessionHandle sessionHandle,
        bool autoSave,
        CancellationToken cancellationToken)
        where TEntity : class, IEntity
    {
        // Your logic here.
    }
}
```

```

## See Also

- \* [Entities] (Entities.md)
- \* [Repositories] (Repositories.md)

## 11.4 Dapper

## # Dapper Integration

[Dapper](<https://github.com/DapperLib/Dapper>) is a simple and lightweight object mapper for .NET. A key feature of Dapper is its [high performance](<https://github.com/DapperLib/Dapper#performance>) compared to other ORMs.

While you can use Dapper as is in your ABP applications, there is also an integration package that simplifies creating repository classes using Dapper.

> ABP's Dapper integration package is based on Entity Framework Core (EF Core). That means it assumes you will use Dapper mixed with EF Core where EF Core is the primary database provider and you use Dapper when you need to fine-tune your queries and get the maximum performance. See [this article](<https://community.abp.io/posts/using-dapper-with-the-abp-framework-shp74p21>) if you want to know why it is like that.

### ## Installation

You can use the [ABP CLI](CLI.md) to install the [Volo.Abp.Dapper](<https://www.nuget.org/packages/Volo.Abp.Dapper>) package to your project. Execute the following command in the folder of the `\*.csproj` file that you want to install the package on:

```
```bash
abp add-package Volo.Abp.Dapper
```
```

> If you haven't done it yet, you first need to install the ABP CLI. For other installation options, see [the package description page](<https://abp.io/package-detail/Volo.Abp.Dapper>).  
>  
> If you have a layered solution, it is suggested to install that package to your database layer of the solution.

### ## Implement a Dapper Repository

The best way to interact with Dapper is to create a [repository](Repositories.md) class that abstracts your Dapper database operations. The following example creates a new repository class that works with the `People` table:

```
```C#
public class PersonDapperRepository :
    DapperRepository<MyAppDbContext>, ITransientDependency
{
    public PersonDapperRepository(IDbContextProvider<MyAppDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }

    public virtual async Task<List<string>> GetAllPersonNamesAsync()
    {
        var dbConnection = await GetDbConnectionAsync();
        return (await dbConnection.QueryAsync<string>(
```

```

        "select Name from People",
        transaction: await GetDbTransactionAsync()
    ).ToList();
}

public virtual async Task<int> UpdatePersonNamesAsync(string name)
{
    var dbConnection = await GetDbConnectionAsync();
    return await dbConnection.ExecuteAsync(
        "update People set Name = @NewName",
        new { NewName = name },
        await GetDbTransactionAsync()
    );
}
```

```

Let's examine this class:

- It inherits from the `DapperRepository` class, which provides useful methods and properties for database operations. It also implements the `IUnitOfWorkEnabled` interface, so ABP makes the database connection (and transaction if requested) available in the method body by implementing dynamic proxies (a.k.a. interception).
- It gets an `IDbContextProvider<MyAppDbContext>` object where `MyAppDbContext` is type of your Entity Framework Core `DbContext` class. It should be configured as explained in the [EF Core document](Entity-Framework-Core.md). If you've created by ABP's startup template, then it should already be configured.
- The `GetAllPersonNamesAsync` and `UpdatePersonNamesAsync` method's been made `virtual`. That's needed to make the interception process working.
- We've used the `GetDbConnectionAsync` and `GetDbTransactionAsync` methods to obtain the current database connection and transaction (that is managed by ABP's [Unit of Work](Unit-Of-Work.md) system).

Then you can [inject](Dependency-Injection.md) `PersonDapperRepository` to any service to perform these database operations. If you want to implement a layered solution, we suggest to introduce an `IPersonDapperRepository` interface in your domain layer, implement it in your database later, then inject the interface to use the repository service.

> If you want to learn more details and examples of using Dapper with the ABP Framework, [check this community article](https://community.abp.io/posts/using-dapper-with-the-abp-framework-shp74p21).

## ## See Also

- \* [Community Article: Using Dapper with the ABP Framework](https://community.abp.io/posts/using-dapper-with-the-abp-framework-shp74p21)
- \* [Entity Framework Core integration document](Entity-Framework-Core.md)

## 12 Real Time

### 12.1 SignalR Integration

#### # SignalR Integration

> It is already possible to follow [the standard Microsoft tutorial](<https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr>) to add [SignalR](<https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction>) to your application. However, ABP provides SignalR integration packages those simplify the integration and usage.

#### ## Installation

##### ### Server Side

It is suggested to use the [ABP CLI](CLI.md) to install this package.

##### #### Using the ABP CLI

Open a command line window in the folder of your project (.csproj file) and type the following command:

```
```bash
abp add-package Volo.Abp.AspNetCore.SignalR
```
```

> You typically want to add this package to the web or API layer of your application, depending on your architecture.

> If you haven't done it yet, you first need to install the [ABP CLI](CLI.md). For other installation options, see [the package description page](<https://abp.io/package-detail/Volo.Abp.AspNetCore.SignalR>).

##### #### Manual Installation

If you want to manually install;

1. Add the [Volo.Abp.AspNetCore.SignalR](<https://www.nuget.org/packages/Volo.Abp.AspNetCore.SignalR>) NuGet package to your project:

```
```
Install-Package Volo.Abp.AspNetCore.SignalR
```
```

Or use the Visual Studio NuGet package management UI to install it.

2. Add the `AbpAspNetCoreSignalRModule` to the dependency list of your module:

```
```csharp
```

```
[DependsOn(
    //...other dependencies
    typeof(AbpAspNetCoreSignalRModule) //Add the new module dependency
)]
public class YourModule : AbpModule
{
}
...
```

> You don't need to use the `services.AddSignalR()` and the `app.UseEndpoints(...)` , it's done by the `AbpAspNetCoreSignalRModule`.

Client Side

Client side installation depends on your UI framework / client type.

ASP.NET Core MVC / Razor Pages UI

Run the following command in the root folder of your web project:

```
```bash
yarn add @abp/signalr
```
```

> This requires to [install yarn](https://yarnpkg.com/) if you haven't install before.

This will add the `@abp/signalr` to the dependencies in the `package.json` of your project:

```
```json
{
 ...
 "dependencies": {
 ...
 "@abp/signalr": "~2.7.0"
 }
}
```

Run the following [ABP CLI](CLI.md) command in the root folder of your web project:

```
```bash
abp install-libs
```
```

This will copy the SignalR JavaScript files into your project:

```
![signal-js-file](images/signal-js-file.png)
```

Finally, add the following code to your page/view to include the `signalr.js` file

```
```xml
```

```
@section scripts {
    <abp-script type="typeof(SignalRBrowserScriptContributor)" />
}
```

It requires to add `@using Volo.Abp.AspNetCore.Mvc.UI.Packages.SignalR` to your page/view.

> You could add the `signalr.js` file in a standard way. But using the `SignalRBrowserScriptContributor` has additional benefits. See the [Client Side Package Management] (UI/AspNetCore/Client-Side-Package-Management.md) and [Bundling & Minification] (UI/AspNetCore/Bundling-Minification.md) documents for details.

That's all. you can use the [SignalR JavaScript API] (<https://docs.microsoft.com/en-us/aspnet/core/signalr/javascript-client>) in your page.

Other UI Frameworks / Clients

Please refer to [Microsoft's documentation] (<https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction>) for other type of clients.

The ABP Framework Integration

This section covers the additional benefits when you use the ABP Framework integration packages.

Hub Route & Mapping

ABP automatically registers all the hubs to the [dependency injection] (Dependency-Injection.md) (as transient) and maps the hub endpoint. So, you don't have to use the `app.UseEndpoints(...)` to map your hubs. Hub route (URL) is determined conventionally based on your hub name.

Example:

```
```csharp
public class MessagingHub : Hub
{
 //...
}
```

The hub route will be `/signalr-hubs/messaging` for the `MessagingHub`:

- Adding a standard `/signalr-hubs/` prefix
- Continue with the \*\*kebab-case\*\* hub name, without the `Hub` suffix.

If you want to specify the route, you can use the `HubRoute` attribute:

```
```csharp
[HubRoute("/my-messaging-hub")]
public class MessagingHub : Hub
{
```

```
//...
}
```

```

### ### AbpHub Base Classes

Instead of the standard `Hub` and `Hub<T>` classes, you can inherit from the `AbpHub` or `AbpHub<T>` which have useful base properties like `CurrentUser`.

Example:

```
```csharp
public class MessagingHub : AbpHub
{
    public async Task SendMessage(string targetUserName, string message)
    {
        var currentUserName = CurrentUser.UserName; //Access to the current user info
        var txt = L["MyText"]; //Localization
    }
}
```

```

While you could inject the same properties into your hub constructor, this way simplifies your hub class.

### ### Manual Registration / Mapping

ABP automatically registers all the hubs to the [dependency injection](Dependency-Injection.md) as a **transient service**. If you want to **disable auto dependency injection** registration for your hub class, just add a `DisableConventionalRegistration` attribute. You can still register your hub class to dependency injection in the `ConfigureServices` method of your module if you like:

```
```csharp
context.Services.AddTransient<MessagingHub>();
```

```

When **you or ABP** register the class to the dependency injection, it is automatically mapped to the endpoint route configuration just as described in the previous sections. You can use `DisableAutoHubMap` attribute if you want to manually map your hub class.

For manual mapping, you have two options:

1. Use the `AbpSignalROptions` to add your map configuration (in the `ConfigureServices` method of your [module](Module-Development-Basics.md)), so ABP still performs the endpoint mapping for your hub:

```
```csharp
Configure<AbpSignalROptions>(options =>
{
    options.Hubs.Add(
        new HubConfig(

```

```

        typeof(MessagingHub), //Hub type
        "/my-messaging/route", //Hub route (URL)
        hubOptions =>
    {
        //Additional options
        hubOptions.LongPolling.PollTimeout = TimeSpan.FromSeconds(30);
    }
)
);
```;

```

This is a good way to provide additional SignalR options.

If you don't want to disable auto hub map, but still want to perform additional SignalR configuration, use the `options.Hubs.AddOrUpdate(...)` method:

```

```csharp
Configure<AbpSignalROptions>(options =>
{
    options.Hubs.AddOrUpdate(
        typeof(MessagingHub), //Hub type
        config => //Additional configuration
    {
        config.RoutePattern = "/my-messaging-hub"; //override the default route
        config.ConfigureActions.Add(hubOptions =>
        {
            //Additional options
            hubOptions.LongPolling.PollTimeout = TimeSpan.FromSeconds(30);
        });
    }
);
```;

```

This is the way you can modify the options of a hub class defined in a depended module (where you don't have the source code access).

2. Change `app.UseConfiguredEndpoints` in the `OnApplicationInitialization` method of your `[module] (Module=Development-Basics.md)` as shown below (added a lambda method as the parameter).

```

```csharp
app.UseConfiguredEndpoints(endpoints =>
{
    endpoints.MapHub<MessagingHub>("/my-messaging-hub", options =>
    {
        options.LongPolling.PollTimeout = TimeSpan.FromSeconds(30);
    });
```;

```

### ### UserIdProvider

ABP implements SignalR's `IUserIdProvider` interface to provide the current user id from the `ICurrentUser` service of the ABP framework (see [the current user service](CurrentUser.md)), so it will be integrated to the authentication system of your application. The implementing class is the `AbpSignalRUserIdProvider`, if you want to change/override it.

### ## Example Application

See the [SignalR Integration Demo](<https://github.com/abpframework/abp-samples/tree/master/SignalRDemo>) as a sample application. It has a simple Chat page to send messages between (authenticated) users.

![signalr-demo-chat](images/signalr-demo-chat.png)

### ## Remarks

ABP Framework doesn't change the SignalR. It works in your ABP Framework based application just like any other ASP.NET Core application.

Refer to the Microsoft's documentation to [host and scale](<https://docs.microsoft.com/en-us/aspnet/core/signalr/scale>) your application, integrate to [Azure](<https://docs.microsoft.com/en-us/aspnet/core/signalr/publish-to-azure-web-app>) or [Redis backplane](<https://docs.microsoft.com/en-us/aspnet/core/signalr/redis-backplane>)... etc.

### ## See Also

- [Microsoft SignalR documentation](<https://docs.microsoft.com/en-us/aspnet/core/signalr/introduction>)
- [Real-Time Messaging In A Distributed Architecture Using ABP, SignalR & RabbitMQ](<https://volosoft.com/blog/RealTime-Messaging-Distributed-Architecture-Abp-SignalR-RabbitMQ>)

## 13 Dapr Integration

### # ABP Dapr Integration

➤ This document assumes that you are already familiar with [Dapr](<https://dapr.io/>) and you want to use it in your ABP based applications.

[Dapr](<https://dapr.io/>) (Distributed Application Runtime) provides APIs that simplify microservice connectivity. It is an open source project that is mainly backed by Microsoft. It is also a CNCF (Cloud Native Computing Foundation) project and trusted by the community.

ABP and Dapr have some intersecting features like service-to-service communication, distributed message bus and distributed locking. However their purposes are totally

different. ABP's goal is to provide an end-to-end developer experience by offering an opinionated architecture and providing the necessary infrastructure libraries, reusable modules and tools to implement that architecture properly. Dapr's purpose, on the other hand, is to provide a runtime to decouple common microservice communication patterns from your application logic.

ABP and Dapr can perfectly work together in the same application. ABP offers some packages to provide better integration where Dapr features intersect with ABP. You can use other Dapr features with no ABP integration packages based on [[its own documentation](#)] (<https://docs.dapr.io/>).

## ## ABP Dapr Integration Packages

ABP provides the following NuGet packages for the Dapr integration:

- \* [[Volo.Abp.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.Dapr>) : The main Dapr integration package. All other packages depend on this package.
- \* [[Volo.Abp.Http.Client.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.Http.Client.Dapr>) : Integration package for ABP's [[dynamic](#)] (./API/Dynamic-CSharp-API-Clients.md) and [[static](#)] (./API/Static-CSharp-API-Clients.md) C# API Client Proxies systems with Dapr's [[service invocation](#)] (<https://docs.dapr.io/developing-applications/building-blocks/service-invocation/service-invocation-overview/>) building block.
- \* [[Volo.Abp.EventBus.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.EventBus.Dapr>) : Implements ABP's distributed event bus with Dapr's [[publish & subscribe](#)] (<https://docs.dapr.io/developing-applications/building-blocks/pubsub/>) building block. With this package, you can send events, but can not receive.
- \*
- [[Volo.Abp.AspNetCore.Mvc.Dapr.EventBus](#)] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.Dapr.EventBus>) : Provides the endpoints to receive events from Dapr's [[publish & subscribe](#)] (<https://docs.dapr.io/developing-applications/building-blocks/pubsub/>) building block. Use this package to send and receive events.
- \*
- [[Volo.Abp.DistributedLocking.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.DistributedLocking.Dapr>) : Uses Dapr's [[distributed lock](#)] (<https://docs.dapr.io/developing-applications/building-blocks/distributed-lock/>) building block for [[distributed locking](#)] (./Distributed-Locking.md) service of the ABP Framework.

In the following sections, we will see how to use these packages to use Dapr in your ABP based solutions.

## ## Basics

### ### Installation

> This section explains how to add [[Volo.Abp.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.Dapr>), the core Dapr integration package to your project. If you are using one of the other Dapr integration packages, you can skip this section since this package will be indirectly added.

Use the ABP CLI to add the [[Volo.Abp.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.Dapr>) NuGet package to your project:

- \* Install the [ABP CLI] (<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed it before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.Dapr` package.
- \* Run the `abp add-package Volo.Abp.Dapr` command.

If you want to do it manually, install the [Volo.Abp.Dapr] (<https://www.nuget.org/packages/Volo.Abp.Dapr>) NuGet package to your project and add ` [DependsOn(typeof(AbpDaprModule))]` to the [ABP module] (./Module-Development-Basics.md) class inside your project.

### ### AbpDaprOptions

`AbpDaprOptions` is the main [options class] (./Options.md) that you can configure the global Dapr settings with. **\*\*All settings are optional and you mostly don't need to configure them.\*\*** If you need, you can configure it in the `ConfigureServices` method of your [module class] (./Module-Development-Basics.md):

```
```csharp
Configure<AbpDaprOptions>(options =>
{
    // ...
});
```

Available properties of the `AbpDaprOptions` class:

- * `HttpEndpoint` (optional): HTTP endpoint that is used while creating a `DaprClient` object. If you don't specify, the default value is used.
- * `GrpcEndpoint` (optional): The gRPC endpoint that is used while creating a `DaprClient` object. If you don't specify, the default value is used.
- * `DaprApiToken` (optional): The [Dapr API token] (<https://docs.dapr.io/operations/security/api-token/>) that is used while sending requests from the application to Dapr. It is filled from the `DAPR_API_TOKEN` environment variable by default (which is set by Dapr once it is configured). See the *Security* section in this document for details.
- * `AppApiToken` (optional): The [App API token] (<https://docs.dapr.io/operations/security/app-api-token/>) that is used to validate requests coming from Dapr. It is filled from the `APP_API_TOKEN` environment variable by default (which is set by Dapr once it is configured). See the *Security* section in this document for details.

Alternatively, you can configure the options in the `Dapr` section of your `appsettings.json` file. Example:

```
```csharp
"Dapr": {
 "HttpEndpoint": "http://localhost:3500/"
}
```

### ### Injecting DaprClient

ABP registers the `DaprClient` class to the [dependency injection](./Dependency-Injection.md) system. So, you can inject and use it whenever you need:

```
```csharp
public class MyService : ITransientDependency
{
    private readonly DaprClient _daprClient;

    public MyService(DaprClient daprClient)
    {
        _daprClient = daprClient;
    }

    public async Task DoItAsync()
    {
        // TODO: Use the injected _daprClient object
    }
}
````
```

Injecting `DaprClient` is the recommended way of using it in your application code. When you inject it, the `IAbpDaprClientFactory` service is used to create it, which is explained in the next section.

### ### IAbpDaprClientFactory

`IAbpDaprClientFactory` can be used to create `DaprClient` or `HttpClient` objects to perform operations on Dapr. It uses `AbpDaprOptions`, so you can configure the settings in a central place.

#### **\*\*Example usages:\*\***

```
```csharp
public class MyService : ITransientDependency
{
    private readonly IAbpDaprClientFactory _daprClientFactory;

    public MyService(IAbpDaprClientFactory daprClientFactory)
    {
        _daprClientFactory = daprClientFactory;
    }

    public async Task DoItAsync()
    {
        // Create a DaprClient object with default options
        DaprClient daprClient = await _daprClientFactory.CreateAsync();

        /* Create a DaprClient object with configuring
         * the DaprClientBuilder object */
        DaprClient daprClient2 = await _daprClientFactory
            .CreateAsync(builder =>

```

```

    {
        builder.UseDaprApiToken("...");  

    });  
  

    // Create an HttpClient object  

    HttpClient httpClient = await _daprClientFactory  

        .CreateHttpClientAsync("target-app-id");  

}
```

```

`CreateHttpClientAsync` method also gets optional `daprEndpoint` and `daprApiToken` parameters.

> ABP uses `IAbpDaprClientFactory` when it needs to create a Dapr client. You can also use Dapr API to create client objects in your application. Using `IAbpDaprClientFactory` is recommended, but not required.

## ## C# API Client Proxies Integration

ABP can [dynamically](../API/Dynamic-CSharp-API-Clients.md) or [statically](../API/Static-CSharp-API-Clients.md) generate proxy classes to invoke your HTTP APIs from a Dotnet client application. It makes perfect sense to consume HTTP APIs in a distributed system. The `[Volo.Abp.Http.Client.Dapr](https://www.nuget.org/packages/Volo.Abp.Http.Client.Dapr)` package configures the client-side proxies system, so it uses Dapr's service invocation building block for the communication between your applications.

### ### Installation

Use the ABP CLI to add the `[Volo.Abp.Http.Client.Dapr](https://www.nuget.org/packages/Volo.Abp.Http.Client.Dapr)` NuGet package to your project (to the client side):

- \* Install the `[ABP CLI](https://docs.abp.io/en/abp/latest/CLI)` if you haven't installed before.
- \* Open a command line (terminal) in the directory of the `.csproj` file you want to add the `Volo.Abp.Http.Client.Dapr` package to.
- \* Run the `abp add-package Volo.Abp.Http.Client.Dapr` command.

If you want to do it manually, install the `[Volo.Abp.Http.Client.Dapr](https://www.nuget.org/packages/Volo.Abp.Http.Client.Dapr)` NuGet package to your project and add `[DependsOn(typeof(AbpHttpClientDaprModule))]` to the `[ABP module](../Module-Development-Basics.md)` class inside your project.

### ### Configuration

Once you install the `[Volo.Abp.Http.Client.Dapr](https://www.nuget.org/packages/Volo.Abp.Http.Client.Dapr)` NuGet package, all you need to do is to configure ABP's remote services option either in `appsettings.json` or using the `AbpRemoteServiceOptions` `[options class](../Options.md)`.

### \*\*Example:\*\*

```

```csharp
{
    "RemoteServices": {
        "Default": {
            "BaseUrl": "http://dapr-httpapi/"
        }
    }
}
```

```

`dapr-httpapi` in this example is the application id of the server application in your Dapr configuration.

The remote service name (`Default` in this example) should match the remote service name specified in the `AddHttpClientProxies` call for dynamic client proxies or the `AddStaticHttpClientProxies` call for static client proxies. Using `Default` is fine if your client communicates to a single server. However, if your client uses multiple servers, you typically have multiple keys in the `RemoteServices` configuration. Once you configure the remote service endpoints as Dapr application ids, it will automatically work and make the HTTP calls through Dapr when you use ABP's client proxy system.

➤ See the [\[dynamic\]\(../API/Dynamic-CSharp-API-Clients.md\)](#) and [\[static\]\(../API/Static-CSharp-API-Clients.md\)](#) client proxy documents for details about the ABP's client proxy system.

## ## Distributed Event Bus Integration

[ABP's distributed event bus](../Distributed-Event-Bus.md) system provides a convenient abstraction to allow applications to communicate asynchronously via events. ABP has integration packages with various distributed messaging systems, like RabbitMQ, Kafka, and Azure. Dapr also has a [\[publish & subscribe building block\]\(https://docs.dapr.io/developing-applications/building-blocks/pubsub/pubsub-overview/\)](#) for the same purpose: distributed messaging / events.

ABP's [\[Volo.Abp.EventBus.Dapr\]\(https://www.nuget.org/packages/Volo.Abp.EventBus.Dapr\)](#) and [\[Volo.Abp.AspNetCore.Mvc.Dapr.EventBus\]\(https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.Dapr.EventBus\)](#) packages make it possible to use the Dapr infrastructure for ABP's distributed event bus.

The [\[Volo.Abp.EventBus.Dapr\]\(https://www.nuget.org/packages/Volo.Abp.EventBus.Dapr\)](#) package can be used by any type of application (e.g., a Console or ASP.NET Core application) to publish events through Dapr. To be able to receive messages (by subscribing to events), you need to have the [\[Volo.Abp.AspNetCore.Mvc.Dapr.EventBus\]\(https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.Dapr.EventBus\)](#) package installed, and your application should be an ASP.NET Core application.

## ### Installation

If your application is an ASP.NET Core application and you want to send and receive events, you need to install the

[[Volo.Abp.AspNetCore.Mvc.Dapr.EventBus](#)] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.Dapr.EventBus>) package as described below:

- \* Install the [[ABP CLI](#)] (<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed it before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.AspNetCore.Mvc.Dapr.EventBus` package to.
- \* Run the `abp add-package Volo.Abp.AspNetCore.Mvc.Dapr.EventBus` command.

If you want to do it manually, install the [[Volo.Abp.AspNetCore.Mvc.Dapr.EventBus](#)] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.Dapr.EventBus>) NuGet package to your project and add `'[DependsOn(typeof(AbpAspNetCoreMvcDaprEventBusModule))]'` to the [[ABP module](#)] (./Module-Development-Basics.md) class inside your project.

> \*\*If you install the [[Volo.Abp.AspNetCore.Mvc.Dapr.EventBus](#)] (<https://www.nuget.org/packages/Volo.Abp.AspNetCore.Mvc.Dapr.EventBus>) package, you don't need to install the [[Volo.Abp.EventBus.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.EventBus.Dapr>) package, because the first one already has a reference to the latter one.\*\*

If your application is not an ASP.NET Core application, you can't receive events from Dapr, at least with ABP's integration packages (see [[Dapr's document](#)] (<https://docs.dapr.io/developing-applications/building-blocks/pubsub/howto-publish-subscribe/>) if you want to receive events in a different type of application). However, you can still publish messages using the [[Volo.Abp.EventBus.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.EventBus.Dapr>) package. In this case, follow the steps below to install that package to your project:

- \* Install the [[ABP CLI](#)] (<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed it before.
- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.EventBus.Dapr` package to.
- \* Run the `abp add-package Volo.Abp.EventBus.Dapr` command.

If you want to do it manually, install the [[Volo.Abp.EventBus.Dapr](#)] (<https://www.nuget.org/packages/Volo.Abp.EventBus.Dapr>) NuGet package to your project and add `'[DependsOn(typeof(AbpEventBusDaprModule))]'` to the [[ABP module](#)] (./Module-Development-Basics.md) class inside your project.

### ### Configuration

You can configure the `AbpDaprEventBusOptions` [[options class](#)] (./Options.md) for Dapr configuration:

```
```csharp
Configure<AbpDaprEventBusOptions>(options =>
{
    options.PubSubName = "pubsub";
});
```
```

Available properties of the `AbpDaprEventBusOptions` class:

- \* `PubSubName` (optional): The `pubsubName` parameter while publishing messages through the `DaprClient.PublishEventAsync` method. Default value: `pubsub`.

### ### The ABP Subscription Endpoints

ABP provides the following endpoints to receive events from Dapr:

- \* `dapr/subscribe`: Dapr uses this endpoint to get a list of subscriptions from the application. ABP automatically returns all the subscriptions for your distributed event handler classes and custom controller actions with the `Topic` attribute.
  - \* `api/abp/dapr/event`: The unified endpoint to receive all the events from Dapr. ABP dispatches the events to your event handlers based on the topic name.
- > **\*\*Since ABP will call `MapSubscribeHandler` internally, you should not manually call it anymore.\*\*** You can use the `app.UseCloudEvents()` middleware in your ASP.NET Core pipeline if you want to support the [CloudEvents](<https://cloudevents.io/>) standard.

### ### Usage

#### #### The ABP Way

You can follow [ABP's distributed event bus documentation](./Distributed-Event-Bus.md) to learn how to publish and subscribe to events in the ABP way. No change required in your application code to use Dapr pub-sub. ABP will automatically subscribe to Dapr for your event handler classes (that implement the `IDistributedEventHandler` interface).

ABP provides `api/abp/dapr/event`

**\*\*Example: Publish an event using the `IDistributedEventBus` service\*\***

```
```csharp
public class MyService : ITransientDependency
{
    private readonly IDistributedEventBus _distributedEventBus;

    public MyService(IDistributedEventBus distributedEventBus)
    {
        _distributedEventBus = distributedEventBus;
    }

    public async Task DoItAsync()
    {
        await _distributedEventBus.PublishAsync(new StockCountChangedEto
        {
            ProductCode = "AT837234",
            NewStockCount = 42
        });
    }
}
````
```

**\*\*Example: Subscribe to an event by implementing the `IDistributedEventHandler` interface\*\***

```
```csharp
public class MyHandler : 
    IDistributedEventHandler<StockCountChangedEto>, 
    ITransientDependency
{
    public async Task HandleEventAsync(StockCountChangedEto eventData)
    {
        var productCode = eventData.ProductCode;
        // ...
    }
}
````
```

See [ABP's distributed event bus documentation](../Distributed-Event-Bus.md) to learn the details.

#### #### Using the Dapr API

In addition to ABP's standard distributed event bus system, you can also use Dapr's API to publish events.

➤ If you directly use the Dapr API to publish events, you may not benefit from ABP's standard distributed event bus features, like the outbox/inbox pattern implementation.

**\*\*Example: Publish an event using `DaprClient`\*\***

```
```csharp
public class MyService : ITransientDependency
{
    private readonly DaprClient _daprClient;

    public MyService(DaprClient daprClient)
    {
        _daprClient = daprClient;
    }

    public async Task DoItAsync()
    {
        await _daprClient.PublishEventAsync(
            "pubsub", // pubsub name
            "StockChanged", // topic name
            new StockCountChangedEto // event data
            {
                ProductCode = "AT837234",
                NewStockCount = 42
            }
        );
    }
}
````
```

```
}
```

## \*\*Example: Subscribe to an event by creating an ASP.NET Core controller\*\*

```
~~~~csharp
public class MyController : AbpController
{
    [HttpPost("/stock-changed")]
    [Topic("pubsub", "StockChanged")]
    public async Task<IActionResult> TestRouteAsync([FromBody] StockCountChangedEto model)
    {
        HttpContext.ValidateDaprAppApiToken();

        // Do something with the event
        return Ok();
    }
}
~~~
```

``HttpContext.ValidateDaprAppApiToken()`` extension method is provided by ABP to check if the request is coming from Dapr. This is optional. You should configure Dapr to send the App API token to your application if you want to enable the validation. If not configured, ``ValidateDaprAppApiToken()`` does nothing. See [\[Dapr's App API Token document\]](#) (<https://docs.dapr.io/operations/security/app-api-token/>) for more information. Also see the `*AbpDaprOptions*` and `*Security*` sections in this document.

See the [\[Dapr documentation\]](#) (<https://docs.microsoft.com/en-us/dotnet/architecture/dapr-for-net-developers/publish-subscribe>) to learn the details of sending & receiving events with the Dapr API.

## ## Distributed Lock

➤ Dapr's distributed lock feature is currently in the Alpha stage and may not be stable yet. It is not suggested to replace ABP's distributed lock with Dapr in that point.

ABP provides a [\[Distributed Locking\]](#) ([./Distributed-Locking.md](#)) abstraction to control access to a shared resource by multiple applications. Dapr also has a [\[distributed lock building block\]](#) (<https://docs.dapr.io/developing-applications/building-blocks/distributed-lock/>). The [\[Volo.Abp.DistributedLocking.Dapr\]](#) (<https://www.nuget.org/packages/Volo.Abp.DistributedLocking.Dapr>) package makes ABP use Dapr's distributed locking system.

## ### Installation

Use the ABP CLI to add the [\[Volo.Abp.DistributedLocking.Dapr\]](#) (<https://www.nuget.org/packages/Volo.Abp.DistributedLocking.Dapr>) NuGet package to your project (to the client side):

\* Install the [\[ABP CLI\]](#) (<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed it before.

- \* Open a command line (terminal) in the directory of the `\*.csproj` file you want to add the `Volo.Abp.DistributedLocking.Dapr` package to.
- \* Run the `abp add-package Volo.Abp.DistributedLocking.Dapr` command.

If you want to do it manually, install the [Volo.Abp.DistributedLocking.Dapr] (<https://www.nuget.org/packages/Volo.Abp.DistributedLocking.Dapr>) NuGet package to your project and add ` [DependsOn(typeof(AbpDistributedLockingDaprModule))]` to the [ABP module] (./Module-Development-Basics.md) class inside your project.

#### #### Configuration

You can use the `AbpDistributedLockDaprOptions` options class in the `ConfigureServices` method of [your module] (./Module-Development-Basics.md) to configure the Dapr distributed lock:

```
```csharp
Configure<AbpDistributedLockDaprOptions>(options =>
{
    options.StoreName = "mystore";
});
```

The following options are available:

- * **`StoreName`** (required): The store name used by Dapr. Lock key names are scoped in the same store. That means different applications can acquire the same lock name in different stores. Use the same store name for the same resources you want to control the access of.
- * `Owner` (optional): The `owner` value used by the `DaprClient.Lock` method. If you don't specify, ABP uses a random value, which is fine in general.
- * `DefaultExpirationTimeout` (optional): Default value of the time after which the lock gets expired. Default value: 2 minutes.

Usage

You can inject and use the `IAbpDistributedLock` service, just like explained in the [Distributed Locking document] (./Distributed-Locking.md).

****Example:****

```
```csharp
public class MyService : ITransientDependency
{
 private readonly IAbpDistributedLock _distributedLock;

 public MyService(IAbpDistributedLock distributedLock)
 {
 _distributedLock = distributedLock;
 }

 public async Task MyMethodAsync()
```

```

{
 await using (var handle =
 await _distributedLock.TryAcquireAsync("MyLockName"))
 {
 if (handle != null)
 {
 // your code that access the shared resource
 }
 }
}
```

```

There are two points we should mention about the `TryAcquireAsync` method, as different from ABP's standard usage:

- * The `timeout` parameter is currently not used (even if you specify it), because Dapr doesn't support waiting to obtain the lock.
- * Dapr uses the expiration timeout system (that means the lock is automatically released after that timeout even if you don't release the lock by disposing the handler). However, ABP's `TryAcquireAsync` method has no such a parameter. Currently, you can set `AbpDistributedLockDaprOptions.DefaultExpirationTimeout` as a global value in your application.

As mentioned first, Dapr's distributed lock feature is currently in the Alpha stage and its API is a candidate to change. You should use it as is if you want, but be ready for the changes in the future. For now, we are recommending to use the [DistributedLock](<https://github.com/madelson/DistributedLock>) library as explained in ABP's [Distributed Locking document]([./Distributed-Locking.md](#)).

Security

If you are using Dapr, most or all the incoming and outgoing requests in your application pass through Dapr. Dapr uses two kinds of API tokens to secure the communication between your application and Dapr.

Dapr API Token

> This token is automatically set by default and generally you don't care about it.

The [Enable API token authentication in Dapr](<https://docs.dapr.io/operations/security/api-token/>) document describes what the Dapr API token is and how it is configured. Please read that document if you want to enable it for your application.

If you enable the Dapr API token, you should send that token in every request to Dapr from your application. `AbpDaprOptions` defines a `DaprApiToken` property as a central point to configure the Dapr API token in your application.

The default value of the `DaprApiToken` property is set from the `DAPR_API_TOKEN` environment variable and that environment variable is set by Dapr when it runs. So, most of the time, you don't need to configure `AbpDaprOptions.DaprApiToken` in your

application. However, if you need to configure (or override) it, you can do in the `ConfigureServices` method of your module class as shown in the following code block:

```
```csharp
Configure<AbpDaprOptions>(options =>
{
 options.DaprApiClient = "...";
});
```
```

Or you can set it in your `appsettings.json` file:

```
```json
"Dapr": {
 "DaprApiClient": "..."
}
```
```

Once you set it, it is used when you inject `DaprClient` or use `IAbpDaprClientFactory`. If you need that value in your application, you can inject `IDaprApiClientProvider` and use its `GetDaprApiClient()` method.

App API Token

> Enabling App API token validation is strongly recommended. Otherwise, for example, any client can directly call your event subscription endpoint, and your application acts like an event has occurred (if there is no other security policy in your event subscription endpoint).

The [Authenticate requests from Dapr using token authentication](<https://docs.dapr.io/operations/security/app-api-token/>) document describes what the App API token is and how it is configured. Please read that document if you want to enable it for your application.

If you enable the App API token, you can validate it to ensure that the request is coming from Dapr. ABP provides useful shortcuts to validate it.

****Example: Validate the App API token in an event handling HTTP API****

```
```csharp
public class MyController : AbpController
{
 [HttpPost("/stock-changed")]
 [Topic("pubsub", "StockChanged")]
 public async Task<IActionResult> TestRouteAsync([FromBody] StockCountChangedEto model)
 {
 // Validate the App API token!
 HttpContext.ValidateDaprAppApiToken();

 // Do something with the event
 return Ok();
 }
}
```

```
}
```

``HttpContext.ValidateDaprAppApiToken()`` is an extension method provided by the ABP Framework. It throws an ``AbpAuthorizationException`` if the token was missing or wrong in the HTTP header (the header name is ``dapr-api-token``). You can also inject ``IDaprAppApiTokenValidator`` and use its methods to validate the token in any service (not only in a controller class).

You can configure ``AbpDaprOptions.AppApiToken`` if you want to set (or override) the App API token value. The default value is set by the ``APP_API_TOKEN`` environment variable. You can change it in the ``ConfigureServices`` method of your module class as shown in the following code block:

```
```csharp
Configure<AbpDaprOptions>(options =>
{
    options.AppApiToken = "...";
});
```
```

Or you can set it in your ``appsettings.json`` file:

```
```json
"Dapr": {
    "AppApiToken": "..."
}
```
```

If you need that value in your application, you can inject ``IDaprApiTokenProvider`` and use its ``GetAppApiToken()`` method.

## ## See Also

- \* [Dapr for .NET Developers] (<https://docs.microsoft.com/en-us/dotnet/architecture/dapr-for-net-developers/>)
- \* [The Official Dapr Documentation] (<https://docs.dapr.io/>)

## 14 Testing

### # Automated Testing

#### ## Introduction

ABP Framework has been designed with testability in mind. There are some different levels of automated testing;

- \* **Unit Tests**: You typically test a single class (or a very few classes together). These tests will be fast. However, you generally need to deal with mocking for the dependencies of your service(s).
- \* **Integration Tests**: You typically test a service, but this time you don't mock the fundamental infrastructure and services to see if they properly working together.
- \* **UI Tests**: You test the UI of the application, just like the users interact with your application.

### **Unit Tests vs Integration Tests**

Integration tests have some significant **advantages** compared to unit tests;

- \* **Easier to write** since you don't work to establish mocking and dealing with the dependencies.
- \* Your test code runs with all the real services and infrastructure (including database mapping and queries), so it is much closer to the **real application test**.

While they have some drawbacks;

- \* They are **slower** compared to unit tests since all the infrastructure is prepared for each test case.
- \* A bug in a service may make multiple test cases broken, so it may be **harder to find the real problem** in some cases.

We suggest to go mixed: Write unit or integration test where it is necessary and you find effective to write and maintain it.

### **The Application Startup Template**

The [\[Application Startup Template\]](#) (Startup-Templates/Application.md) comes with the test infrastructure properly installed and configured for you.

### **The Test Projects**

See the following solution structure in the Visual Studio:

![solution-test-projects](images/solution-test-projects.png)

There are more than one test project, organized by the layers;

- \* `Domain.Tests` is used to test your Domain Layer objects (like [\[Domain Services\]](#) (Domain-Services.md) and [\[Entities\]](#) (Entities.md)).
- \* `Application.Tests` is used to test your Application Layer (like [\[Application Services\]](#) (Application-Services.md)).
- \* `EntityFrameworkCore.Tests` is used to test your custom repository implementations or EF Core mappings (this project will be different if you use another [\[Database Provider\]](#) (Data-Access.md)).
- \* `Web.Tests` is used to test the UI Layer (like Pages, Controllers and View Components). This project does exists only for MVC / Razor Page applications.
- \* `TestBase` contains some classes those are shared/used by the other projects.

> `HttpApi.Client.ConsoleTestApp` is not an automated test application. It is an example Console Application that shows how to consume your HTTP APIs from a .NET Console Application.

The following sections will introduce the base classes and other infrastructure included in these projects.

#### ### The Test Infrastructure

The startup solution has the following libraries already installed;

- \* [xUnit] (<https://xunit.net/>) as the test framework.
- \* [NSubstitute] (<https://nsubstitute.github.io/>) as the mocking library.
- \* [Shouldly] (<https://github.com/shouldly/shouldly>) as the assertion library.

While you are free to replace them with your favorite tools, this document and examples will be base on these tooling.

#### ## The Test Explorer

You can use the Test Explorer to view and run the tests in Visual Studio. For other IDEs, see their own documentation.

#### ### Open the Test Explorer

Open the \*Test Explorer\*, under the \*Tests\* menu, if it is not already open:

![vs-test-explorer] (images/vs-test-explorer.png)

#### ### Run the Tests

Then you can click to the Run All or Run buttons to run the tests. The initial startup template has some sample tests for you:

![vs-startup-template-tests] (images/vs-startup-template-tests.png)

#### ### Run Tests In Parallel

The test infrastructure is compatible to run the tests in parallel. It is **\*\*strongly suggested\*\*** to run all the tests in parallel, which is pretty faster then running them one by one.

To enable it, click to the caret icon near to the settings (gear) button and select the \*Run Tests In Parallel\*.

![vs-run-tests-in-parallel] (images/vs-run-tests-in-parallel.png)

#### ## Unit Tests

For Unit Tests, you don't need to much infrastructure. You typically instantiate your class and provide some pre-configured mocked objects to prepare your object to test.

### ### Classes Without Dependencies

In this simplest case, the class you want to test has no dependencies. In this case, you can directly instantiate your class, call its methods and make your assertions.

#### #### Example: Testing an Entity

Assume that you've an `Issue` [entity] (Entities.md) as shown below:

```
```csharp
using System;
using Volo.Abp.Domain.Entities;

namespace MyProject.Issues
{
    public class Issue : AggregateRoot<Guid>
    {
        public string Title { get; set; }
        public string Description { get; set; }
        public bool IsLocked { get; set; }
        public bool IsClosed { get; private set; }
        public DateTime? CloseDate { get; private set; }

        public void Close()
        {
            IsClosed = true;
            CloseDate = DateTime.UtcNow;
        }

        public void Open()
        {
            if (!IsClosed)
            {
                return;
            }

            if (IsLocked)
            {
                throw new IssueStateException("You can not open a locked issue!");
            }

            IsClosed = true;
            CloseDate = null;
        }
    }
}
````
```

Notice that the `IsClosed` and `CloseDate` properties have private setters to force some business rules by using the `Open()` and `Close()` methods;

- \* Whenever you close an issue, the `CloseDate` should be set to the [current time](Timing.md).
- \* An issue can not be re-opened if it is locked. And if it is re-opened, the `CloseDate` should be set to `null`.

Since the `Issue` entity is a part of the Domain Layer, we should test it in the `Domain.Tests` project. Create an `Issue\_Tests` class inside the `Domain.Tests` project:

```
```csharp
using Shouldly;
using Xunit;

namespace MyProject.Issues
{
    public class Issue_Tests
    {
        [Fact]
        public void Should_Set_The_CloseDate_Whenever_Close_An_Issue()
        {
            // Arrange

            var issue = new Issue();
            issue.CloseDate.ShouldBeNull(); // null at the beginning

            // Act

            issue.Close();

            // Assert

            issue.IsClosed.ShouldBeTrue();
            issue.CloseDate.ShouldNotBeNull();
        }
    }
}```
```

This test follows the AAA (Arrange–Act–Assert) pattern;

- * ****Arrange**** part creates an `Issue` entity and ensures the `CloseDate` is `null` at the beginning.
- * ****Act**** part executes the method we want to test for this case.
- * ****Assert**** part checks if the `Issue` properties are same as we expect to be.

`[Fact]` attribute is defined by the [xUnit](https://xunit.net/) library and marks a method as a test method. `Should...` extension methods are provided by the [Shouldly](https://github.com/shouldly/shouldly) library. You can directly use the `Assert` class of the xUnit, but Shouldly makes it much comfortable and straightforward.

When you execute the tests, you will see that is passes successfully:

```
![issue-first-test](images/issue-first-test.png)
```

Let's add two more test methods:

```
```csharp
[Fact]
public void Should_Allow_To_ReOpen_An_Issue()
{
 // Arrange

 var issue = new Issue();
 issue.Close();

 // Act

 issue.Open();

 // Assert

 issue.IsClosed.ShouldBeFalse();
 issue.CloseDate.ShouldBeNull();
}

[Fact]
public void Should_Not_Allow_To_ReOpen_A_Locked_Issue()
{
 // Arrange

 var issue = new Issue();
 issue.Close();
 issue.IsLocked = true;

 // Act & Assert

 Assert.Throws<IssueStateException>(() =>
 {
 issue.Open();
 });
}
````
```

``Assert.Throws`` checks if the executed code throws a matching exception.

➤ See the xUnit & Shoudly documentations to learn more about these libraries.

Classes With Dependencies

If your service has dependencies and you want to unit test this service, you need to mock the dependencies.

Example: Testing a Domain Service

Assume that you've an `IssueManager` [Domain Service] (Domain-Services.md) that is defined as below:

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp;
using Volo.Abp.Domain.Services;

namespace MyProject.Issues
{
 public class IssueManager : DomainService
 {
 public const int MaxAllowedOpenIssueCountForAUser = 3;

 private readonly IIssueRepository _issueRepository;

 public IssueManager(IIssueRepository issueRepository)
 {
 _issueRepository = issueRepository;
 }

 public async Task AssignToUserAsync(Issue issue, Guid userId)
 {
 var issueCount = await _issueRepository.GetIssueCountOfUserAsync(userId);

 if (issueCount >= MaxAllowedOpenIssueCountForAUser)
 {
 throw new BusinessException(
 code: "IM:00392",
 message: $"You can not assign more" +
 $"than {MaxAllowedOpenIssueCountForAUser} issues to a user!");
 }

 issue.AssignedUserId = userId;
 }
 }
}```
```

`IssueManager` depends on the `IssueRepository` service, that will be mocked in this example.

**\*\*Business Rule\*\*:** The example `AssignToUserAsync` doesn't allow to assign more than 3 (`MaxAllowedOpenIssueCountForAUser` constant) issues to a user. If you want to assign an issue in this case, you first need to unassign an existing issue.

The test case below tries to make a valid assignment:

```
```csharp
using System;
```

```

using System.Threading.Tasks;
using NSubstitute;
using Shouldly;
using Volo.Abp;
using Xunit;

namespace MyProject.Issues
{
    public class IssueManager_Tests
    {
        [Fact]
        public async Task Should_Assign_An_Issue_To_A_User()
        {
            // Arrange

            var userId = Guid.NewGuid();

            var fakeRepo = Substitute.For<IIssueRepository>();
            fakeRepo.GetIssueCountOfUserAsync(userId).Returns(1);

            var issueManager = new IssueManager(fakeRepo);

            var issue = new Issue();

            // Act

            await issueManager.AssignToUserAsync(issue, userId);

            //Assert

            issue.AssignedUserId.ShouldBe(userId);
            await fakeRepo.Received(1).GetIssueCountOfUserAsync(userId);
        }
    }
}

```
* `Substitute.For<IIssueRepository>` creates a mock (fake) object that is passed into the `IssueManager` constructor.
* `fakeRepo.GetIssueCountOfUserAsync(userId).Returns(1)` ensures that the `GetIssueCountOfUserAsync` method of the repository returns `1`.
* `issueManager.AssignToUserAsync` doesn't throw any exception since the repository returns `1` for the currently assigned issue count.
* `issue.AssignedUserId.ShouldBe(userId);` line checks if the `AssignedUserId` has the correct value.
* `await fakeRepo.Received(1).GetIssueCountOfUserAsync(userId);` checks if the `IssueManager` called the `GetIssueCountOfUserAsync` method exactly one time.

```

Let's add a second test to see if it prevents to assign issues to a user more than the allowed count:

```
```csharp
```

```

[Fact]
public async Task Should_Not_Allow_To_Assign_Issues_Over_The_Limit()
{
    // Arrange

    var userId = Guid.NewGuid();

    var fakeRepo = Substitute.For<IIssueRepository>();
    fakeRepo
        .GetIssueCountOfUserAsync(userId)
        .Returns(IssueManager.MaxAllowedOpenIssueCountForAUser);

    var issueManager = new IssueManager(fakeRepo);

    // Act & Assert

    var issue = new Issue();

    await Assert.ThrowsAsync<BusinessException>(async () =>
    {
        await issueManager.AssignToUserAsync(issue, userId);
    });

    issue.AssignedUserId.ShouldBeNull();
    await fakeRepo.Received(1).GetIssueCountOfUserAsync(userId);
}
```

```

› For more information on the mocking, see the [\[NSubstitute\]](#) (<https://nsubstitute.github.io/>) documentation.

It is relatively easy to mock a single dependency. But, when your dependencies grow, it gets harder to setup the test objects and mock all the dependencies. See the *\*Integration Tests\** section that doesn't require mocking the dependencies.

### ### Tip: Share the Test Class Constructor

[\[xUnit\]](#) (<https://xunit.net/>) creates a **\*\*new test class instance\*\*** (`IssueManager\_Tests` for this example) for each test method. So, you can move some *\*Arrange\** code into the constructor to reduce the code duplication. The constructor will be executed for each test case and doesn't affect each other, even if they work in parallel.

**\*\*Example: Refactor the `IssueManager\_Tests` to reduce the code duplication\*\***

```

```csharp
using System;
using System.Threading.Tasks;
using NSubstitute;
using Shouldly;
using Volo.Abp;
using Xunit;

```

```

namespace MyProject.Issues
{
    public class IssueManager_Tests
    {
        private readonly Guid _userId;
        private readonly IIssueRepository _fakeRepo;
        private readonly IssueManager _issueManager;
        private readonly Issue _issue;

        public IssueManager_Tests()
        {
            _userId = Guid.NewGuid();
            _fakeRepo = Substitute.For<IIssueRepository>();
            _issueManager = new IssueManager(_fakeRepo);
            _issue = new Issue();
        }

        [Fact]
        public async Task Should_Assign_An_Issue_To_A_User()
        {
            // Arrange
            _fakeRepo.GetIssueCountOfUserAsync(_userId).Returns(1);

            // Act
            await _issueManager.AssignToUserAsync(_issue, _userId);

            //Assert
            _issue.AssignedUserId.ShouldBe(_userId);
            await _fakeRepo.Received(1).GetIssueCountOfUserAsync(_userId);
        }

        [Fact]
        public async Task Should_Not_Allow_To_Assign_Issues_Over_The_Limit()
        {
            // Arrange
            _fakeRepo
                .GetIssueCountOfUserAsync(_userId)
                .Returns(IssueManager.MaxAllowedOpenIssueCountForAUser);

            // Act & Assert
            await Assert.ThrowsAsync<BusinessException>(async () =>
            {
                await _issueManager.AssignToUserAsync(_issue, _userId);
            });

            _issue.AssignedUserId.ShouldBeNull();
            await _fakeRepo.Received(1).GetIssueCountOfUserAsync(_userId);
        }
    }
}
```

```

› Keep your test code clean to create a maintainable test suite.

## ## Integration Tests

› You can also follow the [web application development tutorial] (Tutorials/Part-1.md) to learn developing a full stack application, including the integration tests.

### ### The Integration Test Infrastructure

ABP Provides a complete infrastructure to write integration tests. All the ABP infrastructure and services will perform in your tests. The application startup template comes with the necessary infrastructure pre-configured for you;

### #### The Database

The startup template is configured to use **\*\*in-memory SQLite\*\*** database for the EF Core (for MongoDB, it uses [Mongo2Go](#) (<https://github.com/Mongo2Go/Mongo2Go>) library). So, all the configuration and queries are performed against a real database and you can even test database transactions.

Using in-memory SQLite database has two main advantages;

- \* It is faster compared to an external DBMS.
- \* It creates a **\*\*new fresh database\*\*** for each test case, so tests doesn't affect each other.

› **\*\*Tip\*\*:** Do not use EF Core's In-Memory database for advanced integration tests. It is not a real DBMS and has many differences in details. For example, it doesn't support transaction and rollback scenarios, so you can't truly test the failing scenarios. On the other hand, In-Memory SQLite is a real DBMS and supports the fundamental SQL database features.

### ### The Seed Data

Writing tests against an empty database is not practical. In most cases, you need to some initial data in the database. For example, if you write a test class that query, update and delete the products, it would be helpful to have a few products in the database before executing the test case.

ABP's [\[Data Seeding\]](#) (Data-Seeding.md) system is a powerful way to seed the initial data. The application startup template has a `*YourProject*TestDataSeedContributor` class in the `.TestBase` project. You can fill it to have an initial data that you can use for each test method.

#### **\*\*Example: Create some Issues as the seed data\*\***

```
```csharp
using System.Threading.Tasks;
using MyProject.Issues;
using Volo.Abp.Data;
using Volo.Abp.DependencyInjection;
```

```

namespace MyProject
{
    public class MyProjectTestDataSeedContributor
        : IDataSeedContributor, ITransientDependency
    {
        private readonly IIssueRepository _issueRepository;

        public MyProjectTestDataSeedContributor(IIssueRepository issueRepository)
        {
            _issueRepository = issueRepository;
        }

        public async Task SeedAsync(DataSeedContext context)
        {
            await _issueRepository.InsertAsync(
                new Issue
                {
                    Title = "Test issue one",
                    Description = "Test issue one description",
                    AssignedUserId = TestData.User1Id
                });
            await _issueRepository.InsertAsync(
                new Issue
                {
                    Title = "Test issue two",
                    Description = "Test issue two description",
                    AssignedUserId = TestData.User1Id
                });
            await _issueRepository.InsertAsync(
                new Issue
                {
                    Title = "Test issue three",
                    Description = "Test issue three description",
                    AssignedUserId = TestData.User1Id
                });
            await _issueRepository.InsertAsync(
                new Issue
                {
                    Title = "Test issue four",
                    Description = "Test issue four description",
                    AssignedUserId = TestData.User2Id
                });
        }
    }
}
```

```

Also created a static class to store the User `Ids`:

```

```csharp
using System;

namespace MyProject
{
    public static class TestData
    {
        public static Guid User1Id = Guid.Parse("41951813-5CF9-4204-8B18-CD765DBCBC9B");
        public static Guid User2Id = Guid.Parse("2DAB4460-C21B-4925-BF41-A52750A9B999");
    }
}
```

```

In this way, we can use these known Issues and the User `Id`'s to perform the tests.

#### ### Example: Testing a Domain Service

`AbpIntegratedTest<T>` class (defined in the [\[Volo.Abp.TestBase\]](#) (<https://www.nuget.org/packages/Volo.Abp.TestBase>) package) is used to write tests integrated to the ABP Framework. `T` is the Type of the root module to setup and initialize the application.

The application startup template has base classes in each test project, so you can derive from these base classes to make it easier.

See the `IssueManager` tests are re-written as integration tests

```

```csharp
using System.Threading.Tasks;
using Shouldly;
using Volo.Abp;
using Xunit;

namespace MyProject.Issues
{
    public class IssueManager_Integration_Tests : MyProjectDomainTestBase
    {
        private readonly IssueManager _issueManager;
        private readonly Issue _issue;

        public IssueManager_Integration_Tests()
        {
            _issueManager = GetRequiredService<IssueManager>();
            _issue = new Issue
            {
                Title = "Test title",
                Description = "Test description"
            };
        }

        [Fact]
        public async Task Should_Not_Allow_To_Assign_Issues_Over_The_Limit()

```

```

    {
        // Act & Assert
        await Assert.ThrowsAsync<BusinessException>(async () =>
        {
            await _issueManager.AssignToUserAsync(_issue, TestData.User1Id);
        });

        _issue.AssignedUserId.ShouldBeNull();
    }

    [Fact]
    public async Task Should_Assign_An_Issue_To_A_User()
    {
        // Act
        await _issueManager.AssignToUserAsync(_issue, TestData.User2Id);

        //Assert
        _issue.AssignedUserId.ShouldBe(TestData.User2Id);
    }
}
```

```

\* First test method assigns the issue to the User 1, which has already assigned to 3 issues in the Data Seed code. So, it throws a `BusinessException`.  
\* Second test method assigns the issue to User 2, which has only 1 issue assigned. So, the method succeeds.

This class typically locates in the `.`Domain.Tests` project since it tests a class located in the `.`Domain` project. It is derived from the `MyProjectDomainTestBase` which is already configured to properly run the tests.

Writing such an integration test class is very straightforward. Another benefit is that you won't need to change the test class later when you add another dependency to the `IssueManager` class.

#### #### Example: Testing an Application Service

Testing an [Application Service](Application-Services.md) is not so different. Assume that you've created an `IssueAppService` as defined below:

```

```csharp
using System.Collections.Generic;
using System.Threading.Tasks;
using Volo.Abp.Application.Services;

namespace MyProject.Issues
{
    public class IssueAppService : ApplicationService, IIssueAppService
    {
        private readonly IIssueRepository _issueRepository;

```

```

public IssueAppService(IIssueRepository issueRepository)
{
    _issueRepository = issueRepository;
}

public async Task<List<IssueDto>> GetListAsync()
{
    var issues = await _issueRepository.GetListAsync();

    return ObjectMapper.Map<List<Issue>, List<IssueDto>>(issues);
}
}
```

```

*(\*assuming you've also defined the `IIssueAppService` and `IssueDto` and created the [object mapping] (Object-To-Object-Mapping.md) between `Issue` and the `IssueDto`)\**

Now, you can write a test class inside the `Application.Tests` project:

```

```csharp
using System.Threading.Tasks;
using Shouldly;
using Xunit;

namespace MyProject.Issues
{
    public class IssueAppService_Tests : MyProjectApplicationTestBase
    {
        private readonly IIssueAppService _issueAppService;

        public IssueAppService_Tests()
        {
            _issueAppService = GetRequiredService<IIssueAppService>();
        }

        [Fact]
        public async Task Should_Get_All_Issues()
        {
            //Act
            var issueDtos = await _issueAppService.GetListAsync();

            //Assert
            issueDtos.Count.ShouldBeGreaterThan(0);
        }
    }
}
```

```

It's that simple. This test method tests everything, including the application service, EF Core mapping, object to object mapping and the repository implementation. In this way, you can fully test the Application Layer and the Domain Layer of your solution.

### ### Dealing with Unit of Work in Integration Tests

ABP's [unit of work](Unit-Of-Work.md) system controls the database connection and transaction management in your application. It seamlessly works while you writing your application code, so you may not aware of it.

In the ABP Framework, all the database operations must be performed inside a unit of work scope. When you test an [application service](Application-Services.md) method, the unit of work scope will be the scope of your application service method. If you are testing a [repository](Repositories.md) method, the unit of work scope will be the scope of your repository method.

In some cases, you may need to manually control the unit of work scope. Consider the following test method:

```
```csharp
public class IssueRepository_Tests : MyProjectDomainTestBase
{
    private readonly IRepository<Issue, Guid> _issueRepository;

    public IssueRepository_Tests()
    {
        _issueRepository = GetRequiredService<IRepository<Issue, Guid>>();
    }

    public async Task Should_Query_By_Title()
    {
        IQueryable<Issue> queryable = await _issueRepository.GetQueryableAsync();
        var issue = queryable.FirstOrDefaultAsync(i => i.Title == "My issue title");
        issue.ShouldNotBeNull();
    }
}
````
```

We are using `\\_issueRepository.GetQueryableAsync` to obtain an `IQueryable<Issue>` object. Then, we are using the `FirstOrDefaultAsync` method to query an issue by its title. The database query is executed at this point, and you get an exception indicating that there is no active unit of work.

To make that test properly working, you should manually start a unit of work scope as shown in the following example:

```
```csharp
public class IssueRepository_Tests : MyProjectDomainTestBase
{
    private readonly IRepository<Issue, Guid> _issueRepository;
    private readonly IUnitOfWorkManager _unitOfWorkManager;

    public IssueRepository_Tests()
    {
        _issueRepository = GetRequiredService<IRepository<Issue, Guid>>();
    }
}
````
```

```

 _unitOfWorkManager = GetRequiredService<IUnitOfWorkManager>();
 }

 public async Task Should_Query_By_Title()
 {
 using (var uow = _unitOfWorkManager.Begin())
 {
 IQueryable<Issue> queryable = await _issueRepository.GetQueryableAsync();
 var issue = queryable.FirstOrDefaultAsync(i => i.Title == "My issue title");
 issue.ShouldNotBeNull();
 await uow.CompleteAsync();
 }
 }
}
```

```

We've used the `IUnitOfWorkManager` service to create a unit of work scope, then called the `FirstOrDefaultAsync` method inside that scope, so we don't have the problem anymore.

➤ Note that we've tested the `FirstOrDefaultAsync` to demonstrate the unit of work problem. Normally, as a good principle, you should write tests only your own code.

Working with DbContext

In some cases, you may want to directly work with the Entity Framework's `DbContext` object to perform database operations in your test methods. In this case, you can use `IDbContextProvider<T>` service to obtain a `DbContext` instance inside a unit of work.

The following example shows how you can create a `DbContext` object in a test method:

```

```csharp
public class MyDbContext_Tests : MyProjectDomainTestBase
{
 private readonly IDbContextProvider<MyProjectDbContext> _dbContextProvider;
 private readonly IUnitOfWorkManager _unitOfWorkManager;

 public IssueRepository_Tests()
 {
 _dbContextProvider = GetRequiredService<IDbContextProvider<MyProjectDbContext>>();
 _unitOfWorkManager = GetRequiredService<IUnitOfWorkManager>();
 }

 public async Task Should_Query_By_Title()
 {
 using (var uow = _unitOfWorkManager.Begin())
 {
 var dbContext = await _dbContextProvider.GetDbContextAsync();
 var issue = await dbContext.Issues.FirstOrDefaultAsync(i => i.Title == "My
issue title");
 issue.ShouldNotBeNull();
 await uow.CompleteAsync();
 }
 }
}
```

```

```
    }  
}  
....
```

Just like we've done in the **Dealing with Unit of Work in Integration Tests** section, we should perform `DbContext` operations inside an active unit of work.

For [MongoDB] (MongoDB.md), you can use the `IMongoDbContextProvider<T>` service to obtain a `DbContext` object and directly use MongoDB APIs in your test methods.

UI Tests

In general, there are two types of UI Tests;

Non Visual Tests

Such tests completely depends on your UI Framework choice;

- * For an MVC / Razor Pages UI, you typically make request to the server, get some HTML and test if some expected DOM elements exist in the returned result.
- * Angular has its own infrastructure and practices to test the components, views and services.

See the following documents to learn Non Visual UI Testing;

- * [Testing in ASP.NET Core MVC / Razor Pages] (UI/AspNetCore/Testing.md)
- * [Testing in Angular] (UI/Angular/Testing.md)
- * [Testing in Blazor] (UI/Blazor/Testing.md)

Visual Tests

Visual Tests are used to interact with the application UI just like a real user does. It fully tests the application, including the visual appearance of the pages and components.

Visual UI Testing is out of the scope for the ABP Framework. There are a lot of tooling in the industry (like [Selenium] (<https://www.selenium.dev/>)) that you can use to test your application's UI.

15 Deployment

Deployment

Deploying an ABP application is not different than deploying any .NET or ASP.NET Core application. You can deploy it to a cloud provider (e.g. Azure, AWS, Google Could) or on-premise server, IIS or any other web server. ABP's documentation doesn't contain much information on deployment. You can refer to your provider's documentation.

However, there are some topics that you should care about when you are deploying your applications. Most of them are general software deployment considerations, but you should

understand how to handle them within your ABP based applications. We've prepared guides for this purpose and we suggest you to read these guides carefully before designing your deployment configuration.

Guides

- * [Configuring for OpenIddict](Configuring-OpenIddict.md) : Notes for some essential configurations for OpenIddict.
- * [Configuring for Production](Configuring-Production.md) : Notes for some essential configurations for production environments.
- * [Optimization for Production](Optimizing-Production.md) : Tips and suggestions for optimizing your application on production environments.
- * [Deploying to a Clustered Environment](Clustered-Environment.md) : Explains how to configure your application when you want to run multiple instances of your application concurrently.
- * [Deploying Distributed / Microservice Solutions](Distributed-Microservice.md) : Deployment notes for solutions consisting of multiple applications and/or services.

15.1 Configuring OpenIddict

Configuring OpenIddict

This document introduces how to configure `OpenIddict` in the `AuthServer` project.

There are different configurations in the `AuthServer` project for `Development` and `Production` environment.

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
 var hostingEnvironment = context.Services.GetHostingEnvironment();

 // Development environment
 if (hostingEnvironment.IsDevelopment())
 {
 PreConfigure<AbpOpenIddictAspNetCoreOptions>(options =>
 {
 // This is default value, you can remove this line.
 options.AddDevelopmentEncryptionAndSigningCertificate = true;
 });
 }

 // Production or Staging environment
 if (!hostingEnvironment.IsDevelopment())
 {
 PreConfigure<AbpOpenIddictAspNetCoreOptions>(options =>
 {
 options.AddDevelopmentEncryptionAndSigningCertificate = false;
 });
 }
}
```

```

PreConfigure<OpenIddictServerBuilder>(builder =>
{
 builder.AddSigningCertificate(GetSigningCertificate(hostingEnvironment));
 builder.AddEncryptionCertificate(GetSigningCertificate(hostingEnvironment));

 //...
});

}

private X509Certificate2 GetSigningCertificate(IWebHostEnvironment hostingEnv)
{
 return new X509Certificate2(Path.Combine(hostingEnv.ContentRootPath,
"authserver.pfx"), "00000000-0000-0000-0000-000000000000");
}
```

```

Development Environment

We've enabled ``AddDevelopmentEncryptionAndSigningCertificate`` by default on development environment, It registers (and generates if necessary) a user-specific development encryption/development signing certificate. This is a certificate used for signing and encrypting the tokens and for ****development environment only****.

``AddDevelopmentEncryptionAndSigningCertificate`` cannot be used in applications deployed on IIS or Azure App Service: trying to use them on IIS or Azure App Service will result in an exception being thrown at runtime (unless the application pool is configured to [load a user profile] (<https://learn.microsoft.com/en-us/iis/manage/configuring-security/application-pool-identities#user-profile>)).

To avoid that, consider creating self-signed certificates and storing them in the X.509 certificates storage of the host machine(s). This is the way we do it in production environment.

Production Environment

We've disabled ``AddDevelopmentEncryptionAndSigningCertificate`` in production environment and tried to setup signing and encrypting certificates using ``authserver.pfx``.

You can use the ``dotnet dev-certs https -v -ep authserver.pfx -p 00000000-0000-0000-0000-000000000000`` command to generate the ``authserver.pfx`` certificate.

> ``00000000-0000-0000-000000000000`` is the password of the certificate, you can change it to any password you want.

> Also, please remember to copy ``authserver.pfx`` to the [Content Root Folder] (<https://learn.microsoft.com/en-us/dotnet/api/microsoft.aspnetcore.hosting.ihostingenvironment.contentrootpath?view=aspnet-core-7.0>) of the ``AuthServer`` website.

15.2 Configuring for Production

Configuring Your Application for Production Environments

ABP Framework has a lot of options to configure and fine-tune its features. They are all explained in their own documents. Default values for these options are pretty well for most of the deployment environments. However, you may need to care about some options based on how you've structured your deployment environment. In this document, we will highlight these kind of options. So, it is highly recommended to read this document in order to not have unexpected behaviors in your system in production.

Distributed Cache Prefix

ABP's [distributed cache infrastructure](./Caching.md) provides an option to set a key prefix for all of your data that is saved into your distributed cache provider. The default value of this option is not set (it is `null`). If you are using a distributed cache server that is shared by different applications, then you can set a prefix value to isolate an application's cache data from others.

```
```csharp
Configure<AbpDistributedCacheOptions>(options =>
{
 options.KeyPrefix = "MyCrmApp";
});
```
```

That's all. ABP, then will add this prefix to all of your cache keys in your application as along as you use ABP's `IDistributedCache<TCacheItem>` or `IDistributedCache<TCacheItem, TKey>` services. See the [Caching documentation](./Caching.md) if you are new to distributed caching.

> ****Warning**:** If you use ASP.NET Core's standard `IDistributedCache` service, it's your responsibility to add the key prefix (you can get the value by injecting `IOptions<AbpDistributedCacheOptions>`). ABP can not do it.

> ****Warning**:** Even if you have never used distributed caching in your own codebase, ABP still uses it for some features. So, you should always configure this prefix if your caching server is shared among multiple systems.

> ****Warning**:** If you are building a microservice system, then you will have multiple applications that share the same distributed cache server. In such systems, all applications (or services) should normally use the same cache prefix, because you want all the applications to use the same cache data to have consistency between them.

> ****Warning**:** Some of ABP's startup templates are pre-configured to set a prefix value for the distributed cache. So, please check your application code if it is already configured.

Distributed Lock Prefix

ABP's [distributed locking infrastructure](../Distributed-Locking.md) provides an option to set a prefix for all the keys you are using in the distributed lock server. The default value of this option is not set (it is `null`). If you are using a distributed lock server that is shared by different applications, then you can set a prefix value to isolate an application's lock from others.

```
```csharp
Configure<AbpDistributedLockOptions>(options =>
{
 options.KeyPrefix = "MyCrmApp";
});
```
```

That's all. ABP, then will add this prefix to all of your keys in your application. See the [Distributed Locking documentation](../Distributed-Locking.md) if you are new to distributed locking.

> ****Warning**:** Even if you have never used distributed locking in your own codebase, ABP still uses it for some features. So, you should always configure this prefix if your distributed lock server is shared among multiple systems.

> ****Warning**:** If you are building a microservice system, then you will have multiple applications that share the same distributed locking server. In such systems, all applications (or services) should normally use the same lock prefix, because you want to globally lock your resources in your system.

> ****Warning**:** Some of ABP's startup templates are pre-configured to set a prefix value for distributed locking. So, please check your application code if it is already configured.

Email Sender

ABP's [Email Sending](../Emailing.md) system abstracts sending emails from your application and module code and allows you to configure the email provider and settings in a single place.

Email service is configured to write email contents to the standard [application log](../Logging.md) in development environment. You should configure the email settings to be able to send emails to users in your production environment.

Please see the [Email Sending](../Emailing.md) document to learn how to configure its settings to really send emails.

> ****Warning**:** If you don't configure the email settings, you will get errors while trying to send emails. For example, the [Account module](../Modules/Account.md)'s *Password Reset* feature sends email to the users to reset their passwords if they forget it.

SMS Sender

ABP's [SMS Sending abstraction](https://docs.abp.io/en/abp/latest/SMS-Sending) provides a unified interface to send SMS to users. However, its implementation is left to you.

Because, typically a paid SMS service is used to send SMS, and ABP doesn't depend on a specific SMS provider.

So, if you are using the `ISmsSender` service, you must implement it yourself, as shown in the following code block:

```
```csharp
public class MySmsSender : ISmsSender, ITransientDependency
{
 public async Task SendAsync(SmsMessage smsMessage)
 {
 // TODO: Send it using your provider...
 }
}
````
```

> [ABP Commercial] (<https://commercial.abp.io/>) provides a [Twilio SMS Module] (<https://docs.abp.io/en/commercial/latest/modules/twilio-sms>) as a pre-built integration with the popular [Twilio] (<https://www.twilio.com/>) platform.

BLOB Provider

If you use ABP's [BLOB Storing] (<https://docs.abp.io/en/abp/latest/Blob-Storing>) infrastructure, you should care about the BLOB provider in your production environment. For example, if you use the [File System] (../Blob-Storing-File-System.md) provider and your application is running in a Docker container, you should configure a volume mapping for the BLOB storage path. Otherwise, your data will be lost when the container is restarted. Also, the File System is not a good provider for production if you have a [clustered deployment] (Clustered-Environment.md) or a microservice system.

Check the [BLOB Storing] (../Blob-Storing.md) document to see all the available BLOB storage providers.

> ****Warning****: Even if you don't directly use the BLOB Storage system, a module you are depending on may use it. For example, ABP Commercial's [File Management] (<https://docs.abp.io/en/commercial/latest/modules/file-management>) module stores file contents, and the [Account] (<https://docs.abp.io/en/commercial/latest/modules/account>) module stores user profile pictures in the BLOB Storage system. So, be careful with the BLOB Storing configuration in production. Note that ABP Commercial uses the [Database Provider] (../Blob-Storing-Database.md) as a pre-configured BLOB storage provider, which works in production without any problem, but you may still want to use another provider.

String Encryption

ABP's [`IStringEncryptionService` Service] (../String-Encryption.md) simply encrypts and decrypts given strings based on a password phrase. You should configure the `AbpStringEncryptionOptions` options for the production with a strong password and keep it as a secret. You can also configure the other properties of those options class. See the following example:

```
```csharp
```

```
Configure<AbpStringEncryptionOptions>(options =>
{
 options.DefaultPassPhrase = "gs5nTT042HAL4it1";
});
```

Note that ABP CLI automatically sets the password to a random value on a new project creation. However, it is stored in the `appsettings.json` file and is generally added to your source control. It is suggested to use [User Secrets](https://learn.microsoft.com/en-us/aspnet/core/security/app-secrets) or [Environment Variables](https://learn.microsoft.com/en-us/aspnet/core/fundamentals/configuration) to set that value.

## ## Logging

ABP uses .NET's standard [Logging services](./Logging.md). So, it is compatible with any logging provider that works with .NET. ABP's startup solution templates come with [Serilog](https://serilog.net/) pre-installed and configured for you. It writes logs to file system and console with the initial configuration. File system is useful for development environment, but it is suggested you to use a different provider for your production environment, like Elasticsearch, database or any other provider that can properly work.

## ## The Swagger UI

ABP's startup solution templates come with [Swagger UI](https://swagger.io/) pre-installed. Swagger is a pretty standard and useful tool to discover and test your HTTP APIs on a built-in UI that is embedded into your application or service. It is typically used in development environment, but you may want to enable it on staging or production environments too.

While you will always secure your HTTP APIs with other techniques (like the [Authorization](./Authorization.md) system), allowing malicious software and people to easily discover your HTTP API endpoint details can be considered as a security problem for some systems. So, be careful while taking the decision of enabling or disabling Swagger for the production environment.

➤ You may also want to see the [ABP Swagger integration](./API/Swagger-Integration.md) document.

## 15.3 Deploying to a Clustered Environment

### # Deploying to a Clustered Environment

This document introduces the topics that you should consider when you are deploying your application to a clustered environment where **multiple instances of your application run concurrently**, and explains how you can deal with these topics in your ABP based application.

- This document is valid regardless you have a monolith application or a microservice solution. The Application term is used for a process. An application can be a monolith web application, a service in a microservice solution, a console application, or another kind of an executable process.
- >
- For example, if you are deploying your application to Kubernetes and configure your application or service to run in multiple pods, then your application or service runs in a clustered environment.

## ## Understanding the Clustered Environment

- You can skip this section if you are already familiar with clustered deployment and load balancers.

### ### Single Instance Deployment

Consider an application deployed as a **single instance**, as illustrated in the following figure:

 [deployment-single-instance] (./images/deployment-single-instance.png)

Browsers and other client applications can directly make HTTP requests to your application. You can put a web server (e.g. IIS or NGINX) between the clients and your application, but you still have a single application instance running in a single server or container. Single-instance configuration is **limited to scale** since it runs in a single server and you are limited with the server's capacity.

### ### Clustered Deployment

**Clustered deployment** is the way of running **multiple instances** of your application **concurrently** in a single or multiple servers. In this way, different instances can serve different requests and you can scale by adding new servers to the system. The following figure shows a typical implementation of clustering using a **load balancer**:

 [deployment-clustered] (./images/deployment-clustered.png)

### ### Load Balancers

[Load balancers] ([https://en.wikipedia.org/wiki/Load\\_balancing\\_\(computing\)](https://en.wikipedia.org/wiki/Load_balancing_(computing))) have a lot of features, but they fundamentally **forward an incoming HTTP request** to an instance of your application and return your response back to the client application.

Load balancers can use different algorithms for selecting the application instance while determining the application instance that is used to deliver the incoming request. **Round Robin** is one of the simplest and most used algorithms. Requests are delivered to the application instances in rotation. First instance gets the first request, second instance gets the second, and so on. It returns to the first instance after all the instances are used, and the algorithm goes like that for the next requests.

### ### Potential Problems

Once multiple instances of your application run in parallel, you should carefully consider the following topics:

- \* Any **state (data) stored in memory** of your application will become a problem when you have multiple instances. A state stored in memory of an application instance may not be available in the next request since the next request will be handled by a different application instance. While there are some solutions (like sticky sessions) to overcome this problem user-basis, it is a **best practice to design your application as stateless** if you want to run it in a cluster, container or/and cloud.
- \* **In-memory caching** is a kind of in-memory state and should not be used in a clustered application. You should use **distributed caching** instead.
- \* You shouldn't store data in the **local file system**. It should be available to all instances of your application. Different application instance may run in different containers or servers and they may not be able to have access to the same file system. You can use a **cloud or external storage provider** as a solution.
- \* If you have **background workers** or **job queue managers**, you should be careful since multiple instances may try to execute the same job or perform the same work concurrently. As a result, you may have the same work done multiple times or you may get a lot of errors while trying to access and change the same resources.

You may have more problems with clustered deployment, but these are the most common ones. ABP has been designed to be compatible with the clustered deployment scenario. The following sections explain what you should do when you are deploying your ABP based application to a clustered environment.

## ## Switching to a Distributed Cache

ASP.NET Core provides different kind of caching features. [[In-memory cache](https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory)] (<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/memory>) stores your objects in the memory of the local server and is only available to the application that stored the object. Non-sticky sessions in a clustered environment should use the [[distributed caching](https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed)] (<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>) except some specific scenarios (for example, you can cache a local CSS file into memory. It is read-only data and it is the same in all application instances. You can cache it in memory for performance reasons without any problem).

[[ABP's Distributed Cache](#)] ([./Caching.md](#)) extends [[ASP.NET Core's distributed cache](#)] (<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>) infrastructure. It works in-memory by default. You should configure an actual distributed cache provider when you want to deploy your application to a clustered environment.

➤ You should configure the cache provider for clustered deployment, even if your application doesn't directly use `IDistributedCache`. Because the ABP Framework and the pre-built [[application modules](#)] ([./Modules/Index.md](#)) are using distributed cache.

ASP.NET Core provides multiple integrations to use as your distributed cache provider, like [[Redis](#)] (<https://redis.io/>) and [[NCache](#)] (<https://www.alachisoft.com/ncache/>). You can follow [[Microsoft's documentation](#)] (<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>) to learn how to use them in your applications.

If you decided to use Redis as your distributed cache provider, **\*\*follow [ABP's Redis Cache Integration document](./Redis-Cache.md)\*\*** for the steps you need to follow to install it into your application and setup your Redis configuration.

> Based on your preferences while creating a new ABP solution, Redis cache might be pre-installed in your solution. For example, if you have selected the *\*Tiered\** option with the MVC UI, Redis cache comes as pre-installed. Because, in this case, you have two applications in your solution and they should use the same cache source to be consistent.

## ## Using a Proper BLOB Storage Provider

If you have used ABP's **[BLOB Storing](./Blob-Storing.md)** feature with the **[File System provider](./Blob-Storing-File-System.md)**, you should use another provider in your clustered environment since the File System provider uses the application's local file system.

The **[Database BLOB provider](./Blob-Storing-Database)** is the easiest way since it uses your application's main database (or another database if you configure) to store BLOBs. However, you should remember that BLOBs are large objects and may quickly increase your database's size.

> **[ABP Commercial](https://commercial.abp.io/)** startup solution templates come with the database BLOB provider as pre-installed, and stores BLOBs in the application's database.

Check the **[BLOB Storing](./Blob-Storing.md)** document to see all the available BLOB storage providers.

## ## Configuring Background Jobs

ABP's **[background job system](./Background-Jobs.md)** is used to queue tasks to be executed in the background. Background job queue is persistent and a queued task is guaranteed to be executed (it is re-tried if it fails).

ABP's default background job manager is compatible with clustered environments. It uses a **[distributed lock](./Distributed-Locking.md)** to ensure that the jobs are executed only in a single application instance at a time. See the *\*Configuring a Distributed Lock Provider\** section below to learn how to configure a distributed lock provider for your application, so the default background job manager properly works in a clustered environment.

If you don't want to use a distributed lock provider, you may go with the following options:

- \* Stop the background job manager (set `AbpBackgroundJobOptions.IsJobExecutionEnabled` to `false`) in all application instances except one of them, so only the single instance executes the jobs (while other application instances can still queue jobs).
- \* Stop the background job manager (set `AbpBackgroundJobOptions.IsJobExecutionEnabled` to `false`) in all application instances and create a dedicated application (maybe a console application running in its own container or a Windows Service running in the background) to execute all the background jobs. This can be a good option if your background jobs consume high system resources (CPU, RAM or Disk), so you can deploy that background application to a dedicated server and your background jobs don't affect your application's performance.

- › If you are using an external background job integration (e.g. [\[Hangfire\]](#)(./Background-Workers-Hangfire.md) or [\[Quartz\]](#)(./Background-Workers-Quartz.md)) instead of the default background job manager, then please refer to your provider's documentation to learn how it should be configured for a clustered environment.

## ## Configuring a Distributed Lock Provider

ABP provides a distributed locking abstraction with an implementation made with the [\[DistributedLock\]](#)(<https://github.com/madelson/DistributedLock>) library. A distributed lock is used to control concurrent access to a shared resource by multiple applications to prevent corruption of the resource because of concurrent writes. The ABP Framework and some pre-built [\[application modules\]](#)(./Modules/Index.md) are using distributed locking for several reasons.

However, the distributed lock system works in-process by default. That means it is not distributed actually, unless you configure a distributed lock provider. So, please follow the [\[distributed lock\]](#)(./Distributed-Locking.md) document to configure a provider for your application, if it is not already configured.

## ## Configuring SignalR

ABP provides [\[SignalR\]](#)(./SignalR-Integration.md) integration packages to simplify integration and usage. SignalR can be used whenever you need to add real-time web functionality (real-time messaging, real-time notification etc.) into your application.

SignalR requires that all HTTP requests for a specific connection be handled (needs to keep track of all its connections) by the same server process. So, when SignalR is running on a clustered environment (with multiple servers) **\*\*"sticky sessions"\*\*** must be used.

If you are considering [\[scaling out\]](#)(<https://learn.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-6.0#scale-out>) your servers and don't want to have inconsistency with the active socket connections, you can use [\[Azure SignalR Service\]](#)(<https://learn.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-6.0#azure-signalr-service>) or [\[Redis backplane\]](#)(<https://learn.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-6.0#redis-backplane>).

- › To learn more about how to host and scale SignalR in a clustered environment, please check the [\[ASP.NET Core SignalR hosting and scaling\]](#)(<https://learn.microsoft.com/en-us/aspnet/core/signalr/scale?view=aspnetcore-6.0>).

## ## Implementing Background Workers

ASP.NET Core provides [\[hosted services\]](#)(<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services>) and ABP provides [\[background workers\]](#)(./Background-Workers.md) to perform tasks in background threads in your application.

If your application has tasks running in the background, you should consider how they will behave in a clustered environment, especially if your background tasks are using the same resources. You should design your background tasks so that they continue to work properly in the clustered environment.

Assume that your background worker in your SaaS application checks user subscriptions and sends emails if their subscription renewal date approaches. If the background task runs in multiple application instances, it is probable to send the same email many times to some users, which will disturb them.

We suggest you to use one of the following approaches to overcome the problem:

- \* Implement your background workers so that they work in a clustered environment without any problem. Using the [distributed lock](../Distributed-Locking.md) to ensure concurrency control is a way of doing that. A background worker in an application instance may handle a distributed lock, so the workers in other application instances will wait for the lock. In this way, only one worker does the actual work, while others wait in idle. If you implement this, your workers run safely without caring about how the application is deployed.
- \* Stop the background workers (set `AbpBackgroundWorkerOptions.IsEnabled` to `false`) in all application instances except one of them, so only the single instance runs the workers.
- \* Stop the background workers (set `AbpBackgroundWorkerOptions.IsEnabled` to `false`) in all application instances and create a dedicated application (maybe a console application running in its own container or a Windows Service running in the background) to execute all the background tasks. This can be a good option if your background workers consume high system resources (CPU, RAM or Disk), so you can deploy that background application to a dedicated server and your background tasks don't affect your application's performance.

## 15.4 Distributed / Microservice Solutions

### # Deploying Distributed / Microservice Solutions

The ABP Framework is designed to consider distributed and microservice systems, where you have multiple applications and/or services communicating internally. All of its features are compatible with distributed scenarios. This document highlights some points you should care about when you deploy your distributed or microservice solution.

#### ## Application Name & Instance Id

ABP provides the `IApplicationInfoAccessor` service that provides the following properties:

- \* `ApplicationName`: A human-readable name for an application. It is a unique value for an application.
- \* `InstanceId`: A random (GUID) value generated by the ABP Framework each time you start the application.

These values are used by the ABP Framework in several places to distinguish the application and the application instance (process) in the system. For example, the [audit logging](../Audit-Logging.md) system saves the `ApplicationName` in each audit log record written by the related application, so you can understand which application has created the audit log entry. So, if your system consists of multiple applications saving audit

logs to a single point, you should be sure that each application has a different `ApplicationName`.

The `ApplicationName` property's value is set automatically from the **\*\*entry assembly's name\*\*** (generally, the project name in a .NET solution) by default, which is proper for most cases, since each application typically has a unique entry assembly name.

There are two ways to set the application name to a different value. In this first approach, you can set the `ApplicationName` property in your application's [configuration](../Configuration.md). The easiest way is to add an `ApplicationName` field to your `appsettings.json` file:

```
```json
{
    "ApplicationName": "Services.Ordering"
}
````
```

Alternatively, you can set `AbpApplicationCreationOptions.ApplicationName` while creating the ABP application. You can find the `AddApplication` or `AddApplicationAsync` call in your solution (typically in the `Program.cs` file), and set the `ApplicationName` option as shown below:

```
```csharp
await builder.AddApplicationAsync<OrderingServiceHttpApiHostModule>(options =>
{
    options.ApplicationName = "Services.Ordering";
})
````
```

## ## Using a Distributed Event Bus

ABP's [Distributed Event Bus](../Distributed-Event-Bus.md) system provides a standard interface to communicate with other applications and services. While the name is "distributed", the default implementation is in-process. That means, your applications / services can not communicate with each other unless you explicitly configure a distributed event bus provider.

If you are building a distributed system, then the applications should communicate through an external distributed messaging server. Please follow the [Distributed Event Bus](../Distributed-Event-Bus.md) document to learn how to install and configure your distributed event bus provider.

> **\*\*Warning\*\*:** Even if you don't use the distributed event bus directly in your application code, the ABP Framework and some of the modules you are using may use it. So, if you are building a distributed system, always configure a distributed event bus provider.

> **\*\*Info\*\*:** [Clustered deployment](Clustered-Environment.md) of a single application is not considered as a distributed system. So, if you only have a single application with multiple instances serving behind a load balancer, a real distributed messaging server may not be needed.

## ## See Also

- \* [Deploying to a Clustered Environment](Clustered-Environment.md)

## 15.5 Optimizing for Production

### # Optimizing Your Application for Production Environments

ABP Framework and the startup solution templates are configured well to get the maximum performance on production environments. However, there are still some points you need to pay attention to in order to optimize your system in production. In this document, we will mention some of these topics.

#### ## Caching Static Contents

The following items are contents that can be cached in the client side (typically in the Browser) or in a CDN server:

- \* **\*\*Static images\*\*** can always be cached. Here, you should be careful that if you change an image, use a different file name, or use a versioning query-string parameter, so the browser (or CDN) understands it's been changed.
- \* **\*\*CSS and JavaScript files\*\***. ABP's [bundling & minification](../UI/AspNetCore/Bundling-Minification.md) system always uses a query-string versioning parameter and a hash value in the files names of the CSS & JavaScript files for the [MVC (Razor Pages)](../UI/AspNetCore/Overall.md) UI. So, you can safely cache these files in the client side or in a CDN server.
- \* **\*\*Application bundle files\*\*** of an [Angular UI](../UI/Angular/Quick-Start.md) application.
- \* **\*\*[Application Localization Endpoint](../API/Application-Localization.md)\*\*** can be cached per culture (it already has a `cultureName` query string parameter) if you don't use dynamic localization on the server-side. ABP Commercial's [Language Management](https://commercial.abp.io/modules/Volo.LanguageManagement) module provides dynamic localization. If you're using it, you can't cache that endpoint forever. However, you can still cache it for a while. Applying dynamic localization text changes to the application can delay for a few minutes, even for a few hours in a real life scenario.

There may be more ways based on your solution structure and deployment environment, but these are the essential points you should consider to client-side cache in a production environment.

#### ## Bundling & Minification for MVC (Razor Pages) UI

ABP's [bundling & minification](../UI/AspNetCore/Bundling-Minification.md) system automatically bundles, minifies and versions your CSS and JavaScript files in production environment. Normally, you don't need to do anything, if you haven't disabled it yourself in your application code. It is important to follow the [bundling & minification](../UI/AspNetCore/Bundling-Minification.md) document and truly use the system to get the maximum optimization.

## ## Background Jobs

ABP's [Background Jobs](./Background-Jobs.md) system provides an abstraction with a basic implementation to enqueue jobs and execute them in a background thread. ABP's Default Background Job Manager may not be enough if you are adding too many jobs to the queue and want them to be executed in parallel by multiple servers with a high performance. If you need these, you should consider to configure a dedicated background job software, like [Hangfire](https://www.hangfire.io/). ABP has a pre-built [Hangfire integration](./Background-Jobs-Hangfire.md), so you can switch to Hangfire without changing your application code.

# 16 Application Modules

## 16.1 Overall

### # Application Modules

ABP is a **modular application framework** which consists of dozens of **NuGet & NPM packages**. It also provides a complete infrastructure to build your own application modules which may have entities, services, database integration, APIs, UI components and so on.

There are **two types of modules**. They don't have any structural difference but are categorized by functionality and purpose:

- \* [Framework modules](https://github.com/abpframework/abp/tree/dev/framework/src): These are **core modules of the framework** like caching, emailing, theming, security, serialization, validation, EF Core integration, MongoDB integration... etc. They do not have application/business functionalities but makes your daily development easier by providing common infrastructure, integration and abstractions.
- \* [Application modules](https://github.com/abpframework/abp/tree/dev/modules): These modules implement specific application/business functionalities like blogging, document management, identity management, tenant management... etc. They generally have their own entities, services, APIs and UI components.

### ## Open Source Application Modules

There are some **free and open source** application modules developed and maintained as a part of the ABP Framework.

- \* [Account](Account.md): Provides UI for the account management and allows user to login/register to the application.
- \* [Audit Logging](Audit-Logging.md): Persists audit logs to a database.
- \* [Background Jobs](Background-Jobs.md): Persist background jobs when using the default background job manager.
- \* [CMS Kit](Cms-Kit/Index.md): A set of reusable *Content Management System* features.
- \* [Docs](Docs.md): Used to create technical documentation website. ABP's [own documentation](https://docs.abp.io) already using this module.
- \* [Feature Management](Feature-Management.md): Used to persist and manage the [features](./Features.md).

- \* **[Identity] (Identity.md)**: Manages organization units, roles, users and their permissions, based on the Microsoft Identity library.
- \* **[IdentityServer]** (IdentityServer.md) : Integrates to IdentityServer4.
- \* **[OpenIddict]** (OpenIddict.md) : Integrates to OpenIddict.
- \* **[Permission Management]** (Permission-Management.md) : Used to persist permissions.
- \* **[Setting Management] (Setting-Management.md)**: Used to persist and manage the [settings] (./Settings.md).
- \* **[Tenant Management]** (Tenant-Management.md) : Manages tenants for a [**multi-tenant**] (./Multi-Tenancy.md) application.
- \* **[Virtual File Explorer]** (Virtual-File-Explorer.md) : Provided a simple UI to view files in [**virtual file system**] (./Virtual-File-System.md).

See [[the GitHub repository](#)] (<https://github.com/abpframework/abp/tree/dev/modules>) for source code of all modules.

## ## Commercial Application Modules

[\[ABP Commercial\]](#) (<https://commercial.abp.io/>) license provides **\*\*additional pre-built application modules\*\*** on top of the ABP framework. See the [[module list](#)] (<https://commercial.abp.io/modules>) provided by the ABP Commercial.

## 16.2 Account

### # Account Module

Account module implements the basic authentication features like **\*\*login\*\***, **\*\*register\*\***, **\*\*forgot password\*\*** and **\*\*account management\*\***.

This module is based on [[Microsoft's Identity library](#)] (<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity>) and the [[Identity Module](#)] (Identity.md). It has [[IdentityServer](#)] (<https://github.com/IdentityServer>) integration (based on the [[IdentityServer Module](#)] (IdentityServer.md)) and [[OpenIddict](#)] (<https://github.com/openiddict>) integration (based on the [[OpenIddict Module](#)] (OpenIddict.md)) to provide **\*\*single sign-on\*\***, access control and other advanced authentication features.

### ## How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as package and get updates easily, or you can include its source code into your solution (see `get-source` [[CLI](#)] (./CLI.md) command) to develop your custom module.

### ### The Source Code

The source code of this module can be accessed [[here](#)] (<https://github.com/abpframework/abp/tree/dev/modules/account>). The source code is licensed with [[MIT](#)] (<https://choosealicense.com/licenses/mit/>), so you can freely use and customize it.

### ## User Interface

This section introduces the main pages provided by this module.

#### ### Login

`/Account/Login` page provides the login functionality.

![account-module-login](../images/account-module-login.png)

Social/external login buttons becomes visible if you setup it. See the *\*Social/External Logins\** section below. Register and Forgot password and links redirect to the pages explained in the next sections.

#### ### Register

`/Account/Register` page provides the new user registration functionality.

![account-module-register](../images/account-module-register.png)

#### ### Forgot Password & Reset Password

`/Account/ForgotPassword` page provides a way of sending password reset link to user's email address. The user then clicks to the link and determines a new password.

![account-module-forgot-password](../images/account-module-forgot-password.png)

#### ### Account Management

`/Account/Manage` page is used to change password and personal information of the user.

![account-module-manage-account](../images/account-module-manage-account.png)

#### ## OpenIddict Integration

[Volo.Abp.Account.Web.OpenIddict] (<https://www.nuget.org/packages/Volo.Abp.Account.Web.OpenIddict>) package provides integration for the [OpenIddict] (<https://github.com/openiddict>). This package comes as installed with the [application startup template] (../Startup-Templates/Application.md). See the [OpenIddict Module] (OpenIddict.md) documentation.

#### ## IdentityServer Integration

[Volo.Abp.Account.Web.IdentityServer] (<https://www.nuget.org/packages/Volo.Abp.Account.Web.IdentityServer>) package provides integration for the [IdentityServer] (<https://github.com/IdentityServer>). This package comes as installed with the [application startup template] (../Startup-Templates/Application.md). See the [IdentityServer Module] (IdentityServer.md) documentation.

#### ## Social/External Logins

The Account Module has already configured to handle social or external logins out of the box. You can follow the ASP.NET Core documentation to add a social/external login provider to your application.

#### #### Example: Facebook Authentication

Follow the [ASP.NET Core Facebook integration document] (<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/facebook-logins>) to support the Facebook login for your application.

#### ##### Add the NuGet Package

Add the [Microsoft.AspNetCore.Authentication.Facebook] (<https://www.nuget.org/packages/Microsoft.AspNetCore.Authentication.Facebook>) package to your project. Based on your architecture, this can be ` `.Web` , ` `.IdentityServer` (for tiered setup) or ` `.Host` project.

#### ##### Configure the Provider

Use the ` `.AddFacebook(...)` extension method in the ` ConfigureServices` method of your [module] (./Module-Development-Basics.md), to configure the client:

```
~~~~csharp
context.Services.AddAuthentication()
    .AddFacebook(facebook =>
{
    facebook.AppId = "...";
    facebook.AppSecret = "...";
    facebook.Scope.Add("email");
    facebook.Scope.Add("public_profile");
});
~~~~
```

➤ It would be a better practice to use the ` appsettings.json` or the ASP.NET Core User Secrets system to store your credentials, instead of a hard-coded value like that. Follow the [Microsoft's document] (<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/social/facebook-logins>) to learn the user secrets usage.

## 16.3 Audit Logging

### # Audit Logging Module

The Audit Logging Module basically implements the ` IAuditingStore` to save the audit log objects to a database.

➤ This document covers only the audit logging module which persists audit logs to a database. See [the audit logging] (./Audit-Logging.md) document for more about the audit logging system.

### ## How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as package and get updates easily, or you can include its source code into your solution (see ``get-source` [CLI](./CLI.md)` command) to develop your custom module.

### ### The Source Code

The source code of this module can be accessed [[here](https://github.com/abpframework/abp/tree/dev/modules/audit-logging)](https://github.com/abpframework/abp/tree/dev/modules/audit-logging). The source code is licensed with [[MIT](https://choosealicense.com/licenses/mit/)](https://choosealicense.com/licenses/mit/), so you can freely use and customize it.

### ## Internals

#### ### Domain Layer

##### #### Aggregates

- ``AuditLog`` (aggregate root): Represents an audit log record in the system.
- ``EntityChange`` (collection): Changed entities of audit log.
- ``AuditLogAction`` (collection): Executed actions of audit log.

##### #### Repositories

Following custom repositories are defined for this module:

- ``IAuditLogRepository``

#### ### Database providers

##### #### Common

##### ##### Table / collection prefix & schema

All tables/collections use the ``Abp`` prefix by default. Set static properties on the ``AbpAuditLoggingDbProperties`` class if you need to change the table prefix or set a schema name (if supported by your database provider).

##### ##### Connection string

This module uses ``AbpAuditLogging`` for the connection string name. If you don't define a connection string with this name, it fallbacks to the ``Default`` connection string. See the [[connection strings](https://docs.abp.io/en/abp/latest/Connection-Strings)](https://docs.abp.io/en/abp/latest/Connection-Strings) documentation for details.

#### ### Entity Framework Core

##### ##### Tables

- **\*\*AbpAuditLogs\*\***
  - `AbpAuditLogActions`
  - `AbpEntityChanges`
    - `AbpEntityPropertyChanges`

#### MongoDB

##### Collections

- **\*\*AbpAuditLogs\*\***

## See Also

\* [Audit logging system](./Audit-Logging.md)

## 16.4 Background Jobs

# Background Jobs Module

The Background Jobs module implements the `IBackgroundJobStore` interface and makes possible to use the default background job manager of the ABP Framework. If you don't want to use this module, then you should implement the `IBackgroundJobStore` interface yourself.

> This document covers only the background jobs module which persists background jobs to a database. See [the background jobs](./Background-Jobs.md) document for more about the background jobs system.

## How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as package and get updates easily, or you can include its source code into your solution (see `get-source` [CLI](./CLI.md) command) to develop your custom module.

### The Source Code

The source code of this module can be accessed [here](https://github.com/abpframework/abp/tree/dev/modules/background-jobs). The source code is licensed with [MIT](https://choosealicense.com/licenses/mit/), so you can freely use and customize it.

## Internals

### Domain Layer

#### Aggregates

- `BackgroundJobRecord` (aggregate root): Represents a background job record.

#### Repositories

Following custom repositories are defined for this module:

- `IBackgroundJobRepository`

### Database providers

#### Common

##### Table / collection prefix & schema

All tables/collections use the `Abp` prefix by default. Set static properties on the `BackgroundJobsDbProperties` class if you need to change the table prefix or set a schema name (if supported by your database provider).

##### Connection string

This module uses `AbpBackgroundJobs` for the connection string name. If you don't define a connection string with this name, it fallbacks to the `Default` connection string. See the [connection strings](https://docs.abp.io/en/abp/latest/Connection-Strings) documentation for details.

#### Entity Framework Core

##### Tables

- **AbpBackgroundJobs**

#### MongoDB

##### Collections

- **AbpBackgroundJobs**

## See Also

\* [Background job system](./Background-Jobs.md)

## 16.5 CMS Kit

# CMS Kit Module

This module provides CMS (Content Management System) capabilities for your application. It provides **core building blocks** and fully working **sub-systems** to create your own website with CMS features enabled, or use the building blocks in your web sites with any purpose.

> **This module currently available only for the MVC / Razor Pages UI\*\*.** While there is no official Blazor package, it can also work in a Blazor Server UI since a Blazor Server UI is actually a hybrid application that runs in an ASP.NET Core MVC / Razor Pages application.

The following features are currently available:

\* Provides a **page** management system to manage dynamic pages with dynamic URLs.

- \* Provides a [\*\*blogging\*\*] (Blogging.md) system to create publish blog posts with multiple blog support.
- \* Provides a [\*\*tagging\*\*] (Tags.md) system to tag any kind of resource, like a blog post.
- \* Provides a [\*\*comment\*\*] (Comments.md) system to add comments feature to any kind of resource, like blog post or a product review page.
- \* Provides a [\*\*reaction\*\*] (Reactions.md) system to add reactions (smileys) feature to any kind of resource, like a blog post or a comment.
- \* Provides a [\*\*rating\*\*] (Ratings.md) system to add rating feature to any kind of resource.
- \* Provides a [\*\*menu\*\*] (Menus.md) system to manage public menus dynamically.
- \* Provides a [\*\*global resources\*\*] (Global-Resources.md) system to add global styles and scripts dynamically.
- \* Provides a [\*\*Dynamic Widget\*\*] (Dynamic-Widget.md) system to create dynamic widgets for page and blog posts.

> You can click on the any feature links above to understand and learn how to use it.

All features are individually usable. If you disable a feature, it completely disappears from your application, even from the database tables, with the help of the [Global Features] (../../Global-Features.md) system.

## ## Pre Requirements

- This module depends on [BlobStoring] (../../Blob-Storing.md) module for keeping media content.
- > Make sure `BlobStoring` module is installed and at least one provider is configured properly. For more information, check the [documentation] (../../Blob-Storing.md).
- CMS Kit uses [distributed cache] (../../Caching.md) for responding faster.
- > Using a distributed cache, such as [Redis] (../../Redis-Cache.md), is highly recommended for data consistency in distributed/clustered deployments.

## ## How to Install

[ABP CLI] (../../CLI.md) allows installing a module to a solution using the `add-module` command. You can install the CMS Kit module in a command-line terminal with the following command:

```
```bash
abp add-module Volo.CmsKit --skip-db-migrations
```

```

> By default, Cms-Kit is disabled by `GlobalFeature`. Because of that the initial migration will be empty. So you can skip the migration by adding `--skip-db-migrations` to command when installing if you are using Entity Framework Core. After enabling Cms-Kit global fureture, please add new migration.

After the installation process, open the `GlobalFeatureConfigurator` class in the `Domain.Shared` project of your solution and place the following code into the `Configure` method to enable all the features in the CMS Kit module.

```
```csharp

```

```
GlobalFeatureManager.Instance.Modules.CmsKit =>
{
    cmsKit.EnableAll();
});
```

Instead of enabling all, you may prefer to enable the features one by one. The following example enables only the [tags] (Tags.md) and [comments] (Comments.md) features:

```
```csharp
GlobalFeatureManager.Instance.Modules.CmsKit =>
{
 cmsKit.Tags.Enable();
 cmsKit.Comments.Enable();
});
```

➤ If you are using Entity Framework Core, do not forget to add a new migration and update your database.

## ## The Packages

This module follows the [module development best practices guide] (<https://docs.abp.io/en/abp/latest/Best-Practices/Index>) and consists of several NuGet and NPM packages. See the guide if you want to understand the packages and relations between them.

CMS kit packages are designed for various usage scenarios. If you check the [CMS kit packages] (<https://www.nuget.org/packages?q=Volo.CmsKit>), you will see that some packages have `Admin` and `Public` suffixes. The reason is that the module has two application layers, considering they might be used in different type of applications. These application layers uses a single domain layer:

- `Volo.CmsKit.Admin.\*` packages contain the functionalities required by admin (back office) applications.
- `Volo.CmsKit.Public.\*` packages contain the functionalities used in public websites where users read blog posts or leave comments.
- `Volo.CmsKit.\*` (without Admin/Public suffix) packages are called as unified packages. Unified packages are shortcuts for adding Admin & Public packages (of the related layer) separately. If you have a single application for administration and public web site, you can use these packages.

## ## Internals

### ### Table / collection prefix & schema

All tables/collections use the `Cms` prefix by default. Set static properties on the `CmsKitDbProperties` class if you need to change the table prefix or set a schema name (if supported by your database provider).

### ### Connection string

This module uses `CmsKit` for the connection string name. If you don't define a connection string with this name, it falls back to the `Default` connection string.

See the [connection strings](https://docs.abp.io/en/abp/latest/Connection-Strings) documentation for details.

## ## Entity Extensions

[Module entity extension](https://docs.abp.io/en/abp/latest/Module-Entity-Extensions) system is a \*\*high-level\*\* extension system that allows you to \*\*define new properties\*\* for existing entities of the dependent modules. It automatically \*\*adds properties to the entity\*\*, \*\*database\*\*, \*\*HTTP API and user interface\*\* in a single point.

To extend entities of the CMS Kit Pro module, open your `YourProjectNameModuleExtensionConfigurator` class inside of your `DomainShared` project and change the `ConfigureExtraProperties` method like shown below.

```
```csharp
public static void ConfigureExtraProperties()
{
    OneTimeRunner.Run(() =>
    {
        ObjectExtensionManager.Instance.Modules()
            .ConfigureCmsKit(cmsKit =>
        {
            cmsKit.ConfigureBlog(plan => // extend the Blog entity
            {
                plan.AddOrUpdateProperty<string>(<!--property type: string
                    "BlogDescription", <!--property name
                    property => {
                        //validation rules
                        property.Attributes.Add(new RequiredAttribute()); //adds required
                        attribute to the defined property

                        //...other configurations for this property
                    }
                );
            });

            cmsKit.ConfigureBlogPost(blogPost => // extend the BlogPost entity
            {
                blogPost.AddOrUpdateProperty<string>(<!--property type: string
                    "BlogPostDescription", <!--property name
                    property => {
                        //validation rules
                        property.Attributes.Add(new RequiredAttribute()); //adds
                        required attribute to the defined property
                        property.Attributes.Add(
                            new StringLengthAttribute(MyConsts.MaximumDescriptionLength) {
                                MinimumLength = MyConsts.MinimumDescriptionLength
                            }
                        );
                );
            });
        });
    });
}
```

```

        //...other configurations for this property
    }
);
});
});
});
}
```


- `ConfigureCmsKit(...)` method is used to configure the entities of the CMS Kit module.
- `cmsKit.ConfigureBlog(...)` is used to configure the **Blog** entity of the CMS Kit module. You can add or update your extra properties of the **Blog** entity.
- `cmsKit.ConfigureBlogPost(...)` is used to configure the **BlogPost** entity of the CMS Kit module. You can add or update your extra properties of the **BlogPost** entity.
- You can also set some validation rules for the property that you defined. In the above sample, `RequiredAttribute` and `StringLengthAttribute` were added for the property named **"BlogPostDescription"**.
- When you define the new property, it will automatically add to **Entity**, **HTTP API** and **UI** for you.
- Once you define a property, it appears in the create and update forms of the related entity.
- New properties also appear in the datatable of the related page.

```

## 16.6 Docs

# Docs Module

## What is Docs Module?

Docs module is an application module for ABP framework. It simplifies software documentation. This module is free and open-source.

### Integration

Currently docs module provides you to store your docs both on GitHub and file system.

### Hosting

Docs module is an application module and does not offer any hosting solution. You can host your docs on-premise or on cloud.

### Versioning

When you use GitHub to store your docs, Docs Module supports versioning. If you have multiple versions for your docs, there will be a combo-box on the UI to switch between versions. If you choose file system to store your docs, it does not support multiple versions.

[The documents](<https://docs.abp.io/>) for ABP framework is also using this module.

➤ Docs module follows the [module architecture best practices]([../Best-Practices/Module-Architecture.md](#)) guide.

## ## Installation

This document covers `Entity Framework Core` provider but you can also select `MongoDB` as your database provider.

### ### 1- Creating an application

If you do not have an existing ABP project, you can either [generate a CLI command from the get started page of the abp.io website](<https://abp.io/get-started>) and runs it or run the command below:

```
```bash
abp new Acme.MyProject
````
```

### ### 2- Running The Empty Application

After you download the project, extract the ZIP file and open `Acme.MyProject.sln`. You will see that the solution consists of `Application`, `Application.Contracts`, `DbMigrator`, `Domain`, `Domain.Shared`, `EntityFrameworkCore`, `HttpApi`, `HttpApi.Client` and `Web` projects. Right click on `Acme.MyProject.Web` project and **Set as StartUp Project**.

![Create a new project]([../images/docs-module\\_solution-explorer.png](#))

The database connection string is located in `appsettings.json` of your `Acme.MyProject.Web` project. If you have a different database configuration, change the connection string.

```
```json
{
  "ConnectionStrings": {
    "Default": "Server=(LocalDb)\\MSSQLLocalDB;Database=MyProject;Trusted_Connection=True"
  }
}
````
```

Run `Acme.MyProject.DbMigrator` project, it will be responsible for applying database migration and seed data. The database `MyProject` will be created in your database server.

Now an empty ABP project has been created! You can now run your project and see the empty website.

To login your website enter `admin` as the username and `1q2w3E\*` as the password.

### ### 3- Installation Module

Docs module packages are hosted on NuGet. There are 4 packages that needs be to installed to your application. Each package has to be installed to the relevant project.

#### #### 3.1- Use ABP CLI

It is recommended to use the ABP CLI to install the module, open the CMD window in the solution file (`.sln`) directory, and run the following command:

```
```bash
abp add-module Volo.Docs
````
```

#### #### 3.2- Manually install

Or you can also manually install nuget package to each project:

\* Install [[Volo.Docs.Domain](#)] (<https://www.nuget.org/packages/Volo.Docs.Domain/>) nuget package to `Acme.MyProject.Domain` project.

```
```bash
Install-Package Volo.Docs.Domain
````
```

\* Install [[Volo.Docs.EntityFrameworkCore](#)] (<https://www.nuget.org/packages/Volo.Docs.EntityFrameworkCore/>) nuget package to `Acme.MyProject.EntityFrameworkCore` project.

```
```bash
Install-Package Volo.Docs.EntityFrameworkCore
````
```

\* Install [[Volo.Docs.Application](#)] (<https://www.nuget.org/packages/Volo.Docs.Application/>) nuget package to `Acme.MyProject.Application` project.

```
```bash
Install-Package Volo.Docs.Application
````
```

\* Install [[Volo.Docs.Web](#)] (<https://www.nuget.org/packages/Volo.Docs.Domain/>) nuget package to `Acme.MyProject.Web` project.

```
```bash
Install-Package Volo.Docs.Web
````
```

#### ##### 3.2.1- Adding Module Dependencies

An ABP module must declare ` `[DependsOn]` attribute if it has a dependency upon another module. Each module has to be added in ` `[DependsOn]` attribute to the relevant project.

\* Open ` `MyProjectDomainModule.cs` and add ` `typeof(DocsDomainModule)` as shown below;

```
```csharp
[DependsOn(
    typeof(DocsDomainModule),
    typeof(AbpIdentityDomainModule),
    typeof(AbpAuditingModule),
    typeof(BackgroundJobsDomainModule),
    typeof(AbpAuditLoggingDomainModule)
)]
public class MyProjectDomainModule : AbpModule
{
    //...
}
```

```

\* Open ` `MyProjectEntityFrameworkCoreModule.cs` and add ` `typeof(DocsEntityFrameworkCoreModule)` as shown below;

```
```csharp
[DependsOn(
    typeof(DocsEntityFrameworkCoreModule),
    typeof(MyProjectDomainModule),
    typeof(AbpIdentityEntityFrameworkCoreModule),
    typeof(AbpPermissionManagementEntityFrameworkCoreModule),
    typeof(AbpSettingManagementEntityFrameworkCoreModule),
    typeof(AbpEntityFrameworkCoreSqlServerModule),
    typeof(BackgroundJobsEntityFrameworkCoreModule),
    typeof(AbpAuditLoggingEntityFrameworkCoreModule)
)]
public class MyProjectEntityFrameworkCoreModule : AbpModule
{
    //...
}
```

```

\* Open ` `MyProjectApplicationModule.cs` and add ` `typeof(DocsApplicationModule)` as shown below;

```
```csharp
[DependsOn(
    typeof(DocsApplicationModule),
    typeof(MyProjectDomainModule),
    typeof(AbpIdentityApplicationModule))]
public class MyProjectApplicationModule : AbpModule
{
    public override void ConfigureServices(ServiceConfigurationContext context)
    {
        Configure<AbpPermissionOptions>(options =>

```

```

    {

options.DefinitionProviders.Add<MyProjectPermissionDefinitionProvider>();
    });

        Configure<AbpAutoMapperOptions>(options =>
    {
        options.AddProfile<MyProjectApplicationAutoMapperProfile>();
    });
}
```

```

\* Open `MyProjectWebModule.cs` and add `typeof(DocsWebModule)` as shown below;

```

```csharp
[DependsOn(
    typeof(DocsWebModule),
    typeof(MyProjectApplicationModule),
    typeof(MyProjectEntityFrameworkCoreModule),
    typeof(AbpAutofacModule),
    typeof(AbpIdentityWebModule),
    typeof(AbpAccountWebModule),
    typeof(AbpAspNetCoreMvcUiBasicThemeModule)
)]
public class MyProjectWebModule : AbpModule
{
    //...
}
```

```

#### ##### 3.2.2- Adding NPM Package

Open `package.json` and add `@abp/docs": "^5.0.0` as shown below:

```

```json
{
    "version": "1.0.0",
    "name": "my-app",
    "private": true,
    "dependencies": {
        "@abp/aspnetcore.mvc.ui.theme.basic": "^5.0.0",
        "@abp/docs": "^5.0.0"
    }
}
```

```

Then open the command line terminal in the `Acme.MyProject.Web` project folder and run the following command:

```

```bash
abp install-libs
```

```

~~~~

#### #### 4- Database Integration

##### ##### 4.1- Entity Framework Integration

If you choose Entity Framework as your database provider, you need to configure the Docs Module. To do this;

- Open `MyProjectMigrationsDbContext.cs` and add `builder.ConfigureDocs()` to the `OnModelCreating()`.

```
```csharp
public class MyProjectMigrationsDbContext : AbpDbContext<MyProjectMigrationsDbContext>
{
    public MyProjectMigrationsDbContext(DbContextOptions<MyProjectMigrationsDbContext>
options)
        : base(options)
    {

    }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        /* Include modules to your migration db context */

        builder.ConfigurePermissionManagement();
        builder.ConfigureSettingManagement();
        builder.ConfigureBackgroundJobs();
        builder.ConfigureAuditLogging();
        builder.ConfigureIdentity();
        builder.ConfigureIdentityServer();
        builder.ConfigureFeatureManagement();
        builder.ConfigureTenantManagement();
        builder.ConfigureDocs(); //Add this line to configure the Docs Module

        /* Configure customizations for entities from the modules included */

        builder.Entity<IdentityUser>(b =>
        {
            b.ConfigureCustomUserProperties();
        });

        /* Configure your own tables/entities inside the ConfigureQaDoc method */

        builder.ConfigureMyProject();
    }
}
```

```

\* Open `Package Manager Console` in `Visual Studio` and choose `Acme.MyProject.EntityFrameworkCore` as default project. Then write the below command to add the migration for Docs Module.

```
```csharp
add-migration Added_Docs_Module
```
```

When the command successfully executes , you will see a new migration file named as `2018122111621\_Added\_Docs\_Module` in the folder `Acme.MyProject.EntityFrameworkCore\Migrations` .

Now, update the database for Docs module database changes. To do this run the below code on `Package Manager Console` in `Visual Studio` . Be sure `Acme.MyProject.EntityFrameworkCore` is still default project.

```
```csharp
update-database
```
```

Finally, you can check your database to see the newly created tables. For example you can see `DocsProjects` table must be added to your database.

#### ### 5- Linking Docs Module

The default route for Docs module is;

```
```txt
/Documents
````
```

To add Docs module link to your application menu;

\* Open `MyProjectMenuContributor.cs` and add the below line to the method `ConfigureMainMenuAsync()` .

```
```csharp
context.Menu.Items.Add(new ApplicationMenuItem("MyProject.Docs", l["Menu:Docs"],
"/Documents"));
```
```

Final look of **\*\*MyProjectMenuContributor.cs\*\***

```
```csharp
private async Task ConfigureMainMenuAsync(MenuConfigurationContext context)
{
    var l =
context.ServiceProvider.GetRequiredService<IStringLocalizer<MyProjectResource>>();

    context.Menu.Items.Insert(0, new ApplicationMenuItem("MyProject.Home",
l["Menu:Home"], "/"));
}
```

```
        context.Menu.Items.Add(new ApplicationMenuItem("MyProject.Docs", 1["Menu:Docs"],  
"/Documents"));  
    }  
}
```

The `Menu:Docs` keyword is a localization key. To localize the menu text, open `Localization\MyProject\en.json` in the project `Acme.MyProject.Domain`. And add the below line

```
```json  
"Menu:Docs": "Documents"
```
```

Final look of **en.json**

```
```json  
{
 "culture": "en",
 "texts": {
 "Menu:Home": "Home",
 "Welcome": "Welcome",
 "LongWelcomeMessage": "Welcome to the application. This is a startup project based on
the ABP framework. For more information, visit abp.io.",
 "Menu:Docs": "Documents"
 }
}
```

The new menu item for Docs Module is added to the menu. Run your web application and browse to `http://localhost:YOUR\_PORT\_NUMBER/documents` URL.

You will see a warning says;

```
```txt  
There are no projects yet!  
```
```

As we have not added any projects yet, this warning is normal.

#### ### 6- Adding New Docs Project

Open `DocsProjects` in your database, and insert a new record with the following field information;

- \* **\*\*Name\*\*:** The display name of the document name which will be shown on the web page.
- \* **\*\*ShortName\*\*:** A short and URL friendly name that will be used in your docs URL.
- \* **\*\*Format\*\*:** The format of the document (for Markdown: `md`, for HTML: `html`)
- \* **\*\*DefaultDocumentName\*\*:** The document for the initial page.
- \* **\*\*NavigationDocumentName\*\*:** The document to be used for the navigation menu (Index).
- \* **\*\*MinimumVersion\*\*:** The minimum version to show the docs. Below version will not be listed.

- \* **DocumentStoreType**: The source of the documents (for GitHub: `GitHub`, for file system `FileSystem`)
- \* **ExtraProperties**: A serialized `JSON` that stores special configuration for the selected `DocumentStoreType`.
- \* **MainWebsiteUrl**: The URL when user clicks to the logo of the Docs module page. You can simply set as `/` to link to your website root address.
- \* **LatestVersionBranchName**: This is a config for GitHub. It's the branch name which to retrieve the docs. You can set it as `master`.

#### #### Sample Project Record for "GitHub"

You can use [ABP Framework] (<https://github.com/abpframework/abp/>) GitHub documents to configure your GitHub document store.

- Name: `ABP framework (GitHub)`
- ShortName: `abp`
- Format: `md`
- DefaultDocumentName: `Index`
- NavigationDocumentName: `docs-nav.json`
- MinimumVersion: `<NULL>` (no minimum version)
- DocumentStoreType: `GitHub`
- ExtraProperties:
 

```
```json
{"GitHubRootUrl":"https://github.com/abpframework/abp/tree/{version}/docs","GitHubAccessToken":"***","GitHubUserAgent":""}
```
```

Note that `GitHubAccessToken` is masked with `\*\*\*`. It's a private token that you must get it from GitHub. See <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/>

- MainWebsiteUrl: `/`
- LatestVersionBranchName: `dev`

For `SQL` databases, you can use the below `T-SQL` command to insert the specified sample into your `DocsProjects` table:

```
```mssql
INSERT [dbo].[DocsProjects] ([Id], [Name], [ShortName], [Format], [DefaultDocumentName], [NavigationDocumentName], [MinimumVersion], [DocumentStoreType], [ExtraProperties], [MainWebsiteUrl], [LatestVersionBranchName], [ParametersDocumentName]) VALUES (N'12f21123-e08e-4f15-bedb-ae0b2d939658', N'ABP framework (GitHub)', N'abp', N'md', N'Index', N'docs-
```

```
nav.json', NULL, N'GitHub',
N'{"GitHubRootUrl":"https://github.com/abpframework/abp/tree/{version}/docs", "GitHubAccessToken": "***", "GitHubUserAgent": ""}', N'/', N'dev', N')
```
```

Be aware that `GitHubAccessToken` is masked. It's a private token and you must get your own token and replace the `\*\*\*` string.

Now you can run the application and navigate to `/Documents`.

#### #### Sample Project Record for "FileSystem"

You can use [ABP Framework](https://github.com/abpframework/abp/) GitHub documents to configure your GitHub document store.

- Name: `ABP framework (FileSystem)`
- ShortName: `abp`
- Format: `md`
- DefaultDocumentName: `Index`
- NavigationDocumentName: `docs-nav.json`
- MinimumVersion: `<NULL>` (no minimum version)
- DocumentStoreType: `FileSystem`
- ExtraProperties:

```
```json
{
  "Path": "C:\\Github\\abp\\docs"
}
```
```

Note that `Path` must be replaced with your local docs directory. You can fetch the ABP Framework's documents from <https://github.com/abpframework/abp/tree/master/docs> and copy to the directory `C:\\Github\\abp\\docs` to get it work.

- MainWebsiteUrl: `/`
- LatestVersionBranchName: `<NULL>`

For `SQL` databases, you can use the below `T-SQL` command to insert the specified sample into your `DocsProjects` table:

```
```mssql
INSERT [dbo].[DocsProjects] ([Id], [Name], [ShortName], [Format], [DefaultDocumentName],
[NavigationDocumentName], [MinimumVersion], [DocumentStoreType], [ExtraProperties],
[MainWebsiteUrl], [LatestVersionBranchName], [ParametersDocumentName]) VALUES (N'12f21123-
e08e-4f15-bedb-ae0b2d939659', N'ABP framework (FileSystem)', N'abp', N'md', N'Index',
```

```
N' docs=nav.json', NULL, N' FileSystem', N' {"Path": "C:\\Github\\abp\\docs"}', N' /', NULL,  
N'')  
~~~
```

Add one of the sample projects above and run the application. In the menu you will see `Documents` link, click the menu link to open the documents page.

So far, we have created a new application from abp.io website and made it up and ready for Docs module.

7- Creating a New Document

In the sample Project records, you see that `Format` is specified as `md` which refers to [Mark Down](https://en.wikipedia.org/wiki/Markdown). You can see the mark down cheat sheet following the below link;

<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

ABP Docs Module can render mark down to HTML.

Now let's have a look a sample document in markdown format.

```
~~~markdown  
# This is a header  
  
Welcome to Docs Module.  
  
## This is a sub header  
  
[This is a link] (https://abp.io)  
  
![This is an image] (https://abp.io/assets/my-image.png)  
  
## This is a code block  
  
```csharp  
public class Person
{
 public string Name { get; set; }

 public string Address { get; set; }
}
~~~
```

As an example you can see ABP Framework documentation:

[<https://github.com/abpframework/abp/blob/master/docs/en/>] (<https://github.com/abpframework/abp/blob/master/docs/en/>)

#### ##### Conditional sections feature (Using Scriban)

Docs module uses [Scriban] (<https://github.com/lunet-io/scriban/tree/master/doc>) for conditionally show or hide some parts of a document. In order to use that feature, you have to create a JSON file as **\*\*Parameter document\*\*** per every language. It will contain all the key-values, as well as their display names.

For example, [en/docs-params.json] (<https://github.com/abpio/abp-commercial-docs/blob/master/en/docs-params.json>) :

```
```json
{
  "parameters": [
    {
      "name": "UI",
      "displayName": "UI",
      "values": {
        "MVC": "MVC / Razor Pages",
        "NG": "Angular"
      }
    },
    {
      "name": "DB",
      "displayName": "Database",
      "values": {
        "EF": "Entity Framework Core",
        "Mongo": "MongoDB"
      }
    },
    {
      "name": "Tiered",
      "displayName": "Tiered",
      "values": {
        "No": "Not Tiered",
        "Yes": "Tiered"
      }
    }
  ]
}
```

```

Since not every single document in your projects may not have sections or may not need all of those parameters, you have to declare which of those parameters will be used for sectioning the document, as a JSON block anywhere on the document.

For example [Getting-Started.md] (<https://github.com/abpio/abp-commercial-docs/blob/master/en/getting-started.md>) :

```
```
.....
```
```json
//[doc-params]
{
  "UI": ["MVC", "NG"],
  "DB": ["EF", "Mongo"],
  ...
}
```

```
        "Tiered": ["Yes", "No"]  
    }  
    ...  
  
.....
```

This section will be automatically deleted during render. And of course, those key values must match with the ones in **Parameter document**.

! [Interface] (./images/docs-section-ui.png)

Now you can use **Scriban** syntax to create sections in your document.

For example:

```
----  
{{ if UI == "NG" }}  
  
* `--u` argument specifies the UI framework, `angular` in this case.  
  
{{ end }}  
  
{{ if DB == "Mongo" }}  
  
* `--d` argument specifies the database provider, `mongodb` in this case.  
  
{{ end }}  
  
{{ if Tiered == "Yes" }}  
  
* `--tiered` argument is used to create N-tiered solution where authentication server, UI  
and API layers are physically separated.  
  
{{ end }}  
  
----
```

You can also use variables in a text, adding **\_Value** postfix to its key:

```
```txt  
This document assumes that you prefer to use **{{ UI_Value }}** as the UI framework and  
**{{ DB_Value }}** as the database provider.  
```
```

Also, **Document\_Language\_Code** and **Document\_Version** keys are pre-defined if you want to get the language code or the version of the current document (This may be useful for creating links that redirects to another documentation system in another domain).

-----

**\*\*IMPORTANT NOTICE\*\*:** Scriban uses “{{” and “}}” for syntax. Therefore, you must use escape blocks if you are going to use those in your document (an Angular document, for example). See [[Scriban docs](https://github.com/lunet-io/scriban/blob/master/doc/language.md#13-escape-block)] (<https://github.com/lunet-io/scriban/blob/master/doc/language.md#13-escape-block>) for more information.

#### ### 8- Creating the Navigation Document

Navigation document is the main menu of the documents page. It is located on the left side of the page. It is a `JSON` file. Take a look at the below sample navigation document to understand the structure.

```
```json
{
  "items": [
    {
      "text": "Sample Menu Item - 1",
      "items": [
        {
          "text": "Sample Menu Item - 1.1",
          "items": [
            {
              "text": "Sample Menu Item - 1.1.1",
              "path": "SampleMenuItem_1_1_1.md"
            }
          ]
        },
        {
          "text": "Sample Menu Item - 1.2",
          "items": [
            {
              "text": "Sample Menu Item - 1.2.1",
              "path": "SampleMenuItem_1_2_1.md"
            },
            {
              "text": "Sample Menu Item - 1.2.2",
              "path": "SampleMenuItem_1_2_2.md"
            }
          ]
        }
      ]
    },
    {
      "text": "Sample Menu Item - 2",
      "items": [
        {
          "text": "Sample Menu Item - 2.1",
          "items": [
            {
              "text": "Sample Menu Item - 2.1.1",
              "path": "SampleMenuItem_2_1_1.md"
            }
          ]
        }
      ]
    }
  ]
}
```

```

        }
    ]
}
```

```

The upper sample `JSON` file renders the below navigation menu as `HTML`.

```
![Navigation menu](../images/docs-module_download-sample-navigation-menu.png)
```

Finally a new Docs Module is added to your project which is feeded with GitHub.

## ## Full-Text Search(Elastic Search)

The Docs module supports full-text search using Elastic Search. It is not enabled by default. You can configure `DocsElasticSearchOptions` to enable it.

```
```csharp
Configure<DocsElasticSearchOptions>(options =>
{
    options.Enable = true;
    options.IndexName = "your_index_name"; //default IndexName is abp_documents
});
```

The `Index` is automatically created after the application starts if the `Index` does not exist.

`DefaultElasticClientProvider` is responsible for creating `IElasticClient`. By default, it reads Elastic Search's `Url` from `IConfiguration`. If your `IElasticClient` needs additional configuration, please use override `IElasticClientProvider` service and replace it in the [dependency injection](./Dependency-Injection.md) system.

```
```json
{
    "ElasticSearch": {
        "Url": "http://localhost:9200"
    }
}
```

## ## Row Highlighting

You can apply highlight to specific code lines or a range of sequential lines. See the following examples:

```
```
```C# {3, 5}
```

## Next

Docs Module is also available as a standalone application. Check out [\[VoloDocs\]](#)(../../../../Apps/VoloDocs).

## 16.7 Feature Management

## # Feature Management Module

The Feature Management module implements the `IFeatureManagementStore` interface defined by the [\[Feature System\]](#) ([..../Features.md](#)).

> This document covers only the feature management module which persists feature values to a database. See [[the features](#)] (./Features.md) document for more about the feature system.

## ## How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as package and get updates easily, or you can include its source code into your solution (see ``get-source` [CLI](./CLI.md)` command) to develop your custom module.

### ### The Source Code

The source code of this module can be accessed [[here](https://github.com/abpframework/abp/tree/dev/modules/feature-management)](<https://github.com/abpframework/abp/tree/dev/modules/feature-management>). The source code is licensed with [[MIT](https://choosealicense.com/licenses/mit/)](<https://choosealicense.com/licenses/mit/>), so you can freely use and customize it.

## ## User Interface

### ### Feature Management Dialog

Feature management module provides a reusable dialog to manage features related to an object. For example, the [[Tenant Management Module](#)](Tenant-Management.md) uses it to manage features of tenants in the Tenant Management page.

![features-module-opening](./images/features-module-opening.png)

When you click *\*Actions\* -> \*Features\** for a tenant, the feature management dialog is opened. An example screenshot from this dialog with two features defined:

![features-modal](./images/features-modal.png)

In this dialog, you can enable, disable or set values for the features for a tenant.

## ## IFeatureManager

``IFeatureManager`` is the main service provided by this module. It is used to read and change the setting values for the tenants in a multi-tenant application. ``IFeatureManager`` is typically used by the *\*Feature Management Dialog\**. However, you can inject it if you need to set a feature value.

› If you just want to read feature values, use the ``IFeatureChecker`` as explained in the [[Features document](#)](./Features.md).

### \*\*Example: Get/set a feature's value for a tenant\*\*

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.FeatureManagement;

namespace Demo
{
    public class MyService : ITransientDependency
    {
        private readonly IFeatureManager _featureManager;
```

```

public MyService(IFeatureManager featureManager)
{
    _featureManager = featureManager;
}

public async Task SetFeatureDemoAsync(Guid tenantId, string value)
{
    await _featureManager
        .SetForTenantAsync(tenantId, "Feature1", value);

    var currentValue = await _featureManager
        .GetOrDefaultForTenantAsync("Feature1", tenantId);
}
}
```

```

## ## Feature Management Providers

Features Management Module is extensible, just like the [features system](../Features.md). You can extend it by defining feature management providers. There are 3 pre-built feature management providers registered in the following order:

- \* `DefaultValueFeatureManagementProvider`: Gets the value from the default value of the feature definition. It can not set the default value since default values are hard-coded on the feature definition.
  - \* `EditionFeatureManagementProvider`: Gets or sets the feature values for an edition. Edition is a group of features assigned to tenants. Edition system has not implemented by the Tenant Management module. You can implement it yourself or purchase the ABP Commercial [SaaS Module](https://commercial.abp.io/modules/Volo.SaaS) which implements it and also provides more SaaS features, like subscription and payment.
  - \* `TenantFeatureManagementProvider`: Gets or sets the features values for tenants.
- `IFeatureManager` uses these providers on get/set methods. Typically, every feature management provider defines extension methods on the `IFeatureManager` service (like `SetForTenantAsync` defined by the tenant feature management provider).

If you want to create your own provider, implement the `IFeatureManagementProvider` interface or inherit from the `FeatureManagementProvider` base class:

```

```csharp
public class CustomFeatureProvider : FeatureManagementProvider
{
    public override string Name => "Custom";

    public CustomFeatureProvider(IFeatureManagementStore store)
        : base(store)
    {
    }
}
```

```

`FeatureManagementProvider` base class makes the default implementation (using the `IFeatureManagementStore`) for you. You can override base methods as you need. Every provider must have a unique name, which is `Custom` in this example (keep it short since it is saved to database for each feature value record).

Once you create your provider class, you should register it using the `FeatureManagementOptions` [options class] (../Options.md) :

```
```csharp
Configure<FeatureManagementOptions>(options =>
{
    options.Providers.Add<CustomFeatureProvider>();
});
```
```

The order of the providers are important. Providers are executed in the reverse order. That means the `CustomFeatureProvider` is executed first for this example. You can insert your provider in any order in the `Providers` list.

## ## See Also

- \* [Features] (../Features.md)

## 16.8 Identity

### # Identity Management Module

Identity module is used to manage roles, users and their permissions, based on the [Microsoft Identity library] (<https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity>).

### ## How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as package and get updates easily, or you can include its source code into your solution (see `get-source` [CLI] (../CLI.md) command) to develop your custom module.

### ### The Source Code

The source code of this module can be accessed [here] (<https://github.com/abpframework/abp/tree/dev/modules/identity>). The source code is licensed with [MIT] (<https://choosealicense.com/licenses/mit/>), so you can freely use and customize it.

### ## User Interface

This module provides [Blazor] (../UI/Blazor/Overall.md), [Angular] (../UI/Angular/Quick-Start.md) and [MVC / Razor Pages] (../UI/AspNetCore/Overall.md) UI options.

### ### Menu Items

This module adds an *\*Identity management\** menu item under the *\*Administration\** menu:

![identity-module-menu](./images/identity-module-menu.png)

The menu items and the related pages are authorized. That means the current user must have the related permissions to make them visible. The `admin` role (and the users with this role - like the `admin` user) already has these permissions. If you want to enable permissions for other roles/users, open the *\*Permissions\** dialog on the *\*Roles\** or *\*Users\** page and check the permissions as shown below:

![identity-module-permissions](./images/identity-module-permissions.png)

See the [\[Authorization document\]](#)(./Authorization.md) to understand the permission system.

### ### Pages

This section introduces the main pages provided by this module.

#### #### Users

This page is used to see the list of users. You can create/edit and delete users, assign users to roles.

![identity-module-users](./images/identity-module-users.png)

A user can have zero or more roles. Users inherit permissions from their roles. In addition, you can assign permissions directly to the users (by clicking the *\*Actions\** button, then selecting the *\*Permissions\**).

#### #### Roles

Roles are used to group permissions assign them to users.

![identity-module-roles](./images/identity-module-roles.png)

Beside the role name, there are two properties of a role:

- \* `Default`: If a role is marked as "default", then that role is assigned to new users by default when they register to the application themselves (using the [\[Account Module\]](#)(Account.md)).
- \* `Public`: A public role of a user can be seen by other users in the application. This feature has no usage in the Identity module, but provided as a feature that you may want to use in your own application.

## ## Other Features

This section covers some other features provided by this module which don't have the UI pages.

### ### Organization Units

Organization Units (OU) can be used to **\*\*hierarchically group users and entities\*\***.

#### #### OrganizationUnit Entity

An OU is represented by the **\*\*OrganizationUnit\*\*** entity. The fundamental properties of this entity are:

- **\*\*TenantId\*\***: Tenant's Id of this OU. Can be null for host OUs.
- **\*\*ParentId\*\***: Parent OU's Id. Can be null if this is a root OU.
- **\*\*Code\*\***: A hierarchical string code that is unique for a tenant.
- **\*\*DisplayName\*\***: Shown name of the OU.

#### #### Organization Tree

Since an OU can have a parent, all OUs of a tenant are in a **\*\*tree\*\*** structure. There are some rules for this tree;

- There can be more than one root (where the `ParentId` is `null`).
- There is a limit for the first-level children count of an OU (because of the fixed OU Code unit length explained below).

#### #### OU Code

OU code is automatically generated and maintained by the ``OrganizationUnitManager`` service. It's a string that looks something like this:

**```\*\*00001.00042.00005\*\*```**

This code can be used to easily query the database for all the children of an OU (recursively). There are some rules for this code (automatically applied when you use ``OrganizationUnitManager``):

- It is **\*\*unique\*\*** for a [tenant] (./Multi-Tenancy.md).
- All the children of the same OU have codes that **\*\*start with the parent OU's code\*\***.
- It's **\*\*fixed length\*\*** and based on the level of the OU in the tree, as shown in the sample.
- While the OU code is unique, it can be **\*\*changed\*\*** if you move the related OU.

Notice that you must reference an OU by Id, not Code, because the Code can be changed later.

#### #### OrganizationUnit Manager

The ``OrganizationUnitManager`` class can be [injected] (./Dependency-Injection.md) and used to manage OUs. Common use cases are:

- Create, Update or Delete an OU
- Move an OU in the OU tree.
- Getting information about the OU tree and its items.

#### ## Identity Security Log

The security log system records some important operations or changes about your account (like *\*login\** and *\*change password\**). You can also save the security log if needed.

You can inject and use `IdentitySecurityLogManager` or `ISecurityLogManager` to write security logs. It will create a log object by default and fill in some common values, such as `CreationTime`, `ClientIpAddress`, `BrowserInfo`, `current user/tenant`, etc. Of course, you can override them.

```
```cs
await IdentitySecurityLogManager.SaveAsync(new IdentitySecurityLogContext()
{
    Identity = "IdentityServer",
    Action = "ChangePassword"
});
```

Configure `AbpSecurityLogOptions` to provide the application name (in case of you have multiple applications and want to distinguish the applications in the logs) for the log or disable this feature.

```
```cs
Configure<AbpSecurityLogOptions>(options =>
{
    options.ApplicationName = "AbpSecurityTest";
});
```
## Options
```

`IdentityOptions` is the standard [options class](./Options.md) provided by the Microsoft [Identity library](https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity). So, you can set these options in the `ConfigureServices` method of your [module](./Module-Development-Basics.md) class.

#### **\*\*Example: Set minimum required length of passwords\*\***

```
```csharp
Configure<IdentityOptions>(options =>
{
    options.Password.RequiredLength = 5;
});
```

ABP takes these options one step further and allows you to change them on runtime by using the [setting system](./Settings.md). You can [inject](./Dependency-Injection.md) `ISettingManager` and use one of the `Set...` methods to change the option values for a user, a tenant or globally for all users.

#### **\*\*Example: Change minimum required length of passwords for the current tenant\*\***

```
```csharp
```

```

public class MyService : ITransientDependency
{
    private readonly ISettingManager _settingManager;

    public MyService(ISettingManager settingManager)
    {
        _settingManager = settingManager;
    }

    public async Task ChangeMinPasswordLength(int minLength)
    {
        await _settingManager.SetForCurrentTenantAsync(
            IdentitySettingNames.Password.RequiredLength,
            minLength.ToString()
        );
    }
}
```

```

`IdentitySettingNames` class (in the `Volo.Abp.Identity.Settings` namespace) defines constants for the setting names.

#### ## Distributed Events

This module defines the following ETOs (Event Transfer Objects) to allow you to subscribe to changes on the entities of the module;

- \* `UserEto` is published on changes done on an `IdentityUser` entity.
- \* `IdentityRoleEto` is published on changes done on an `IdentityRole` entity.
- \* `IdentityClaimTypeEto` is published on changes done on an `IdentityClaimType` entity.
- \* `OrganizationUnitEto` is published on changes done on an `OrganizationUnit` entity.

#### **\*\*Example: Get notified when a new user has been created\*\***

```

````csharp
public class MyHandler :
    IDistributedEventHandler<EntityCreatedEto<UserEto>>,
    ITransientDependency
{
    public async Task HandleEventAsync(EntityCreatedEto<UserEto> eventData)
    {
        UserEto user = eventData.Entity;
        // TODO: ...
    }
}
```

```

`UserEto` and `IdentityRoleEto` are configured to automatically publish the events. You should configure yourself for the others. See the [Distributed Event Bus document](../Distributed-Event-Bus.md) to learn details of the pre-defined events.

➤ Subscribing to the distributed events is especially useful for distributed scenarios (like microservice architecture). If you are building a monolithic application, or listening events in the same process that runs the Identity Module, then subscribing to the [local events](../Local-Event-Bus.md) can be more efficient and easier.

## ## Internals

This section covers some internal details of the module that you don't need much, but may need to use in some cases.

### ### Domain layer

#### #### Aggregates

##### ##### User

A user is generally a person logins to and uses the application.

- \* `IdentityUser` (aggregate root): Represents a user in the system.
  - \* `IdentityUserRole` (collection): Roles to the user.
  - \* `IdentityUserClaim` (collection): Custom claims of the user.
  - \* `IdentityUserLogin` (collection): External logins of the user.
  - \* `IdentityUserToken` (collection): Tokens of the user (used by the Microsoft Identity services).

##### ##### Role

A role is typically a group of permissions to assign to the users.

- \* `IdentityRole` (aggregate root): Represents a role in the system.
  - \* `IdentityRoleClaim` (collection): Custom claims of the role.

##### ##### Claim Type

A claim type is a definition of a custom claim that can be assigned to other entities (like roles and users) in the system.

- \* `IdentityClaimType` (aggregate root): Represents a claim type definition. It contains some properties (e.g. Required, Regex, Description, ValueType) to define the claim type and the validation rules.

##### ##### Identity Security Log

A `IdentitySecurityLog` object represents an authentication related operation (like \*login\*) in the system.

- \* `IdentitySecurityLog` (aggregate root): Represents a security log in the system.

##### ##### OrganizationUnit

An Organization unit is a entity in a hierarchical structure.

- \* ```OrganizationUnit``` (aggregate root): Represents an organization unit in the system.
- \* ```Roles``` (collection): Roles of the organization unit.

#### #### Repositories

Following custom repositories are defined for this module:

- \* `IIdentityUserRepository`
- \* `IIdentityRoleRepository`
- \* `IIdentityClaimTypeRepository`
- \* ```IIdentitySecurityLogRepository```
- \* ```IOrganizationUnitRepository```

#### #### Domain services

##### ##### User manager

`IdentityUserManager` is used to manage users, their roles, claims, passwords, emails, etc. It is derived from Microsoft Identity's `UserManager<T>` class where `T` is `IdentityUser`.

##### ##### Role manager

`IdentityRoleManager` is used to manage roles and their claims. It is derived from Microsoft Identity's `RoleManager<T>` class where `T` is `IdentityRole`.

##### ##### Claim type manager

`IdentityClaimTypeManager` is used to perform some operations for the `IdentityClaimType` aggregate root.

##### ##### Organization unit manager

```OrganizationUnitManager``` is used to perform some operations for the ```OrganizationUnit``` aggregate root.

##### ##### Security log manager

```IdentitySecurityLogManager``` is used to save security logs.

#### ## Application Layer

#### #### Application Services

- \* `IdentityUserAppService` (implements `IIdentityUserAppService`): Implements the use cases of the user management UI.
- \* `IdentityRoleAppService` (implements `IIdentityRoleAppService`): Implements the use cases of the role management UI.
- \* `IdentityClaimTypeAppService` (implements `IIdentityClaimTypeAppService`): Implements the use cases of the claim type management UI.
- \* `IdentitySettingsAppService` (implements `IIdentitySettingsAppService`): Used to get and update settings for the Identity module.

- \* `IdentityUserLookupAppService` (implements `IIdentityUserLookupAppService`): Used to get information for a user by `id` or `userName`. It is aimed to be used internally by the ABP framework.
- \* `ProfileAppService` (implements `IProfileAppService`): Used to change a user's profile and the password.
- \* `IdentitySecurityLogAppService` (implements `IIdentitySecurityLogAppService`): Implements the use cases of the security logs UI.
- \* `OrganizationUnitAppService` (implements `OrganizationUnitAppService`): Implements the use cases of the organization unit management UI.

#### #### Database Providers

This module provides [[Entity Framework Core](#)] (./Entity-Framework-Core.md) and [[MongoDB](#)] (./MongoDB.md) options for the database.

#### ##### Entity Framework Core

[[Volo.Abp.Identity.EntityFrameworkCore](#)] (<https://www.nuget.org/packages/Volo.Abp.Identity.EntityFrameworkCore>) NuGet package implements the EF Core integration.

#### ##### Database Tables

- \* **AbpRoles**
- \* AbpRoleClaims
- \* **AbpUsers**
- \* AbpUserClaims
- \* AbpUserLogins
- \* AbpUserRoles
- \* AbpUserTokens
- \* **AbpClaimTypes**
- \* **AbpOrganizationUnits**
- \* AbpOrganizationUnitRoles
- \* AbpUserOrganizationUnits
- \* **AbpSecurityLogs**

#### #### MongoDB

[[Volo.Abp.Identity.MongoDB](#)] (<https://www.nuget.org/packages/Volo.Abp.Identity.MongoDB>) NuGet package implements the MongoDB integration.

#### ##### Database Collections

- \* **AbpRoles**
- \* **AbpUsers**
- \* **AbpClaimTypes**
- \* **AbpOrganizationUnits**
- \* **AbpSecurityLogs**

#### #### Common Database Properties

You can set the following properties of the `AbpIdentityDbProperties` class to change the database options:

- \* `DbTablePrefix` (`Abp` by default) is the prefix for table/collection names.
- \* `DbSchema` (`null` by default) is the database schema.
- \* `ConnectionStringName` (`AbpIdentity` by default) is the [connection string](./Connection-Strings.md) name for this module.

These are static properties. If you want to set, do it in the beginning of your application (typically, in `Program.cs`).

## 16.9 IdentityServer

### # IdentityServer Module

IdentityServer module provides a full integration with the [\[IdentityServer4\]](#) (<https://github.com/IdentityServer/IdentityServer4>) (IDS) framework, which provides advanced authentication features like single sign-on and API access control. This module persists clients, resources and other IDS-related objects to database. **\*\*This module is replaced by\*\*** [\[OpenIddict module\]](#) (<https://docs.abp.io/en/abp/latest/Modules/OpenIddict>) after ABP v6.0 in the startup templates.

➤ Note: You can not use IdentityServer and OpenIddict modules together. They are separate OpenID provider libraries for the same job.

### ## How to Install

You don't need this module when you are using OpenIddict module. However, if you want to keep using IdentityServer4 for your applications, you can install this module and remove the OpenIddict module. You can continue to use it as package and get updates easily, or you can include its source code into your solution (see `get-source` [CLI](./CLI.md) command) to develop your custom module.

### ### The Source Code

The source code of this module can be accessed [\[here\]](#) (<https://github.com/abpframework/abp/tree/dev/modules/identityserver>). The source code is licensed with [\[MIT\]](#) (<https://choosealicense.com/licenses/mit/>), so you can freely use and customize it.

### ## User Interface

This module implements the domain logic and database integrations, but not provides any UI. Management UI is useful if you need to add clients and resources on the fly. In this case, you may build the management UI yourself or consider to purchase the [\[ABP Commercial\]](#) (<https://commercial.abp.io/>) which provides the management UI for this module.

### ## Relations to Other Modules

This module is based on the [Identity Module](Identity.md) and have an [integration package](<https://www.nuget.org/packages/Volo.Abp.Account.Web.IdentityServer>) with the [Account Module](Account.md).

## Options

#### AbpIdentityServerBuilderOptions

`AbpIdentityServerBuilderOptions` can be configured in `PreConfigureServices` method of your Identity Server [module](<https://docs.abp.io/en/abp/latest/Module-Development-Basics>). Example:

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
    PreConfigure<AbpIdentityServerBuilderOptions>(builder =>
    {
        //Set options here...
    });
}
```
```

`AbpIdentityServerBuilderOptions` properties:

- \* `UpdateJwtSecurityTokenHandlerDefaultInboundClaimTypeMap` (default: true): Updates `JwtSecurityTokenHandler.DefaultInboundClaimTypeMap` to be compatible with Identity Server claims.
- \* `UpdateAbpClaimTypes` (default: true): Updates `AbpClaimTypes` to be compatible with identity server claims.
- \* `IntegrateToAspNetIdentity` (default: true): Integrate to ASP.NET Identity.
- \* `AddDeveloperSigningCredential` (default: true): Set false to suppress `AddDeveloperSigningCredential()` call on the `IIdentityServerBuilder`.

`IIdentityServerBuilder` can be configured in `PreConfigureServices` method of your Identity Server [module](<https://docs.abp.io/en/abp/latest/Module-Development-Basics>). Example:

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
    PreConfigure<IIdentityServerBuilder>(builder =>
    {
        builder.AddSigningCredential(...);
    });
}
```
```

## Internals

#### Domain Layer

##### Aggregates

## ##### ApiResource

API Resources are needed for allowing clients to request access tokens.

- \* `ApiResource` (aggregate root): Represents an API resource in the system.
- \* `ApiSecret` (collection): secrets of the API resource.
- \* `ApiScope` (collection): scopes of the API resource.
- \* `ApiResourceClaim` (collection): claims of the API resource.

## ##### Client

Clients represent applications that can request tokens from your Identity Server.

- \* `Client` (aggregate root): Represents an Identity Server client application.
- \* `ClientScope` (collection): Scopes of the client.
- \* `ClientSecret` (collection): Secrets of the client.
- \* `ClientGrantType` (collection): Grant types of the client.
- \* `ClientCorsOrigin` (collection): CORS origins of the client.
- \* `ClientRedirectUri` (collection): redirect URIs of the client.
- \* `ClientPostLogoutRedirectUri` (collection): Logout redirect URIs of the client.
- \* `ClientIdPRestriction` (collection): Provider restrictions of the client.
- \* `ClientClaim` (collection): Claims of the client.
- \* `ClientProperty` (collection): Custom properties of the client.

## ##### PersistedGrant

Persisted Grants stores AuthorizationCodes, RefreshTokens and UserConsent.

- \* `PersistedGrant` (aggregate root): Represents PersistedGrant for identity server.

## ##### IdentityResource

Identity resources are data like user ID, name, or email address of a user.

- \* `IdentityResource` (aggregate root): Represents an Identity Server identity resource.
- \* `IdentityClaim` (collection): Claims of identity resource.

## #### Repositories

Following custom repositories are defined for this module:

- \* `IApiResourceRepository`
- \* `IClientRepository`
- \* `IPersistentGrantRepository`
- \* `IIdentityResourceRepository`

## #### Domain Services

This module doesn't contain any domain service but overrides the services below;

- \* `AbpProfileService` (Used when `AbpIdentityServerBuilderOptions.IntegrateToAspNetIdentity` is true)
- \* `AbpClaimsService`
- \* `AbpCorsPolicyService`

#### ### Settings

This module doesn't define any settings.

#### ### Application Layer

##### #### Application Services

- \* `ApiResourceAppService` (implements `IApiResourceAppService`): Implements the use cases of the API resource management UI.
- \* `IdentityServerClaimTypeAppService` (implement `IIdentityServerClaimTypeAppService`): Used to get list of claims.
- \* `ApiResourceAppService` (implements `IApiResourceAppService`): Implements the use cases of the API resource management UI.
- \* `IdentityResourceAppService` (implements `IIdentityResourceAppService`): Implements the use cases of the Identity resource management UI.

#### ### Database Providers

##### #### Common

###### ##### Table/Collection Prefix & Schema

All tables/collections use the `IdentityServer` prefix by default. Set static properties on the `AbpIdentityServerDbProperties` class if you need to change the table prefix or set a schema name (if supported by your database provider).

###### ##### Connection String

This module uses `AbpIdentityServer` for the connection string name. If you don't define a connection string with this name, it fallbacks to the `Default` connection string.

See the [connection strings] (<https://docs.abp.io/en/abp/latest/Connection-Strings>) documentation for details.

#### #### Entity Framework Core

##### ##### Tables

- \* **IdentityServerApiResources**
  - \* IdentityServerApiSecrets
  - \* IdentityServerApiScopes
    - \* IdentityServerApiScopeClaims
  - \* IdentityServerApiClaims
- \* **IdentityServerClients**
  - \* IdentityServerClientScopes
  - \* IdentityServerClientSecrets

```
* IdentityServerClientGrantTypes
* IdentityServerClientCorsOrigins
* IdentityServerClientRedirectUris
* IdentityServerClientPostLogoutRedirectUris
* IdentityServerClientIdPRestrictions
* IdentityServerClientClaims
* IdentityServerClientProperties
* IdentityServerPersistedGrants
* IdentityServerIdentityResources
    * IdentityServerIdentityClaims
```

#### MongoDB

##### Collections

```
* IdentityServerApiResources
* IdentityServerClients
* IdentityServerPersistedGrants
* IdentityServerIdentityResources
```

### 16.9.1 IdentityServer Migration Guide

# Migrating from OpenIddict to IdentityServer4 Step by Step Guide

ABP startup templates use `OpenIddict` OpenID provider from v6.0.0 by default and `IdentityServer` projects are renamed to `AuthServer` in tiered/separated solutions. Since OpenIddict is the default OpenID provider library for ABP templates since v6.0, you may want to keep using [IdentityServer4] (<https://github.com/IdentityServer/IdentityServer4>) library, even it is **archived and no longer maintained by the owners**. ABP doesn't provide support for newer versions of IdentityServer. This guide provides layer-by-layer guidance for migrating your existing [OpenIddict] (<https://github.com/openiddict/openiddict-core>) application to IdentityServer4.

## IdentityServer4 Migration Steps

Use the `abp update` command to update your existing application. See [Upgrading docs] (./Upgrading.md) for more info. Apply required migrations by following the [Migration Guides] (Index.md) based on your application version.

### Domain Shared Layer

- In **MyApplication.Domain.Shared.csproj** replace **project reference**:

```
```csharp
<PackageReference Include="Volo.Abp.OpenIddict.Domain.Shared" Version="6.0.*" />
```
```

with

```
```csharp
```

```
<PackageReference Include="Volo.Abp.IdentityServer.Domain.Shared" Version="6.0.*" />
```

```

- In **\*\*MyApplicationDomainSharedModule.cs\*\*** replace usings and **\*\*module dependencies:\*\***

```
```csharp
using Volo.Abp.OpenIddict;
...
typeof(AbpOpenIddictDomainSharedModule)
```

```

with

```
```csharp
using Volo.Abp.IdentityServer;
...
typeof(AbpIdentityServerDomainSharedModule)
```

```

#### #### Domain Layer

- In **\*\*MyApplication.Domain.csproj\*\*** replace **\*\*project references\*\*:**

```
```csharp
<PackageReference Include="Volo.Abp.OpenIddict.Domain" Version="6.0.*" />
<PackageReference Include="Volo.Abp.PermissionManagement.Domain.OpenIddict"
Version="6.0.*" />
```

```

with

```
```csharp
<PackageReference Include="Volo.Abp.IdentityServer.Domain" Version="6.0.*" />
<PackageReference Include="Volo.Abp.PermissionManagement.Domain.IdentityServer"
Version="6.0.*" />
```

```

- In **\*\*MyApplicationDomainModule.cs\*\*** replace usings and **\*\*module dependencies:\*\***

```
```csharp
using Volo.Abp.OpenIddict;
using Volo.Abp.PermissionManagement.OpenIddict;
...
typeof(AbpOpenIddictDomainModule),
typeof(AbpPermissionManagementDomainOpenIddictModule),
```

```

with

```
```csharp
using Volo.Abp.IdentityServer;
using Volo.Abp.PermissionManagement.IdentityServer;
```

```

```
...
typeof(AbpIdentityServerDomainModule),
typeof(AbpPermissionManagementDomainIdentityServerModule),
```
```

#### #### OpenIddictDataSeedContributor

DataSeeder is the most important part for starting the application since it seeds the initial data for both OpenID providers.

- Create a folder named *\*IdentityServer\** under the Domain project and copy the [\[IdentityServerDataSeedContributor.cs\]](https://github.com/abpframework/abp-samples/blob/master/Ids20OpenId/src/Ids20OpenId.Domain/IdentityServer/IdentityServerDataSeedContributor.cs) (<https://github.com/abpframework/abp-samples/blob/master/Ids20OpenId/src/Ids20OpenId.Domain/IdentityServer/IdentityServerDataSeedContributor.cs>) under this folder. **\*\*Rename\*\*** all the `OpenId2Ids` with your project name.
- Delete *\*OpenIddict\** folder that contains `OpenIddictDataSeedContributor.cs` which is no longer needed.

#### ### EntityFrameworkCore Layer

If you are using MongoDB, skip this step and check the *\*MongoDB\** layer section.

- In **\*\*MyApplication.EntityFrameworkCore.csproj\*\*** replace **\*\*project reference\*\***:

```
```csharp
<PackageReference Include="Volo.Abp.OpenIddict.EntityFrameworkCore" Version="6.0.*" />
```
```

with

```
```csharp
<PackageReference Include="Volo.Abp.IdentityServer.EntityFrameworkCore" Version="6.0.*" />
```
```

- In **\*\*MyApplicationEntityFrameworkCoreModule.cs\*\*** replace usings and **\*\*module dependencies\*\***:

```
```csharp
using Volo.Abp.OpenIddict.EntityFrameworkCore;
...
typeof(AbpOpenIddictEntityFrameworkCoreModule),
```
```

with

```
```csharp
using Volo.Abp.IdentityServer.EntityFrameworkCore;
...
typeof(AbpIdentityServerEntityFrameworkCoreModule),
```
```

- In **\*\*MyApplicationDbContext.cs\*\*** replace usings and **\*\*fluent api configurations\*\***:

```

```csharp
using Volo.Abp.OpenIddict.EntityFrameworkCore;
...
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    /* Include modules to your migration db context */

    ...
    builder.ConfigureOpenIddict();
}
```

```

with

```

```csharp
using Volo.Abp.IdentityServer.EntityFrameworkCore;
...
using Volo.Abp.OpenIddict.EntityFrameworkCore;
...
protected override void OnModelCreating(ModelBuilder builder)
{
    base.OnModelCreating(builder);

    /* Include modules to your migration db context */

    ...
    builder.ConfigureIdentityServer();
}
```

```

➤ Not: You need to create new migration after updating the fluent api. Navigate to *\*EntityFrameworkCore\** folder and add a new migration. Ex, `dotnet ef migrations add Updated\_To\_IdentityServer`

### ### MongoDB Layer

If you are using EntityFrameworkCore, skip this step and check the *\*EntityFrameworkCore\** layer section.

- In **\*\*MyApplication.MongoDB.csproj\*\*** replace **\*\*project reference\*\***:

```

```csharp
<PackageReference Include="Volo.Abp.OpenIddict.MongoDB" Version="6.0.*" />
```

```

with

```

```csharp
<PackageReference Include="Volo.Abp.IdentityServer.MongoDB" Version="6.0.*" />
```

```

- In **\*\*MyApplicationMongoDbModule.cs\*\*** replace usings and **\*\*module dependencies\*\***:

```
```csharp
using Volo.Abp.OpenIddict.MongoDB;
...
typeof(AbpOpenIddictMongoDbModule),
```

```

with

```
```csharp
using Volo.Abp.IdentityServer.MongoDB;
...
typeof(AbpIdentityServerMongoDbModule),
```

```

#### #### DbMigrator Project

- In `appsettings.json` **\*\*replace OpenIddict section with IdentityServer\*\*** since IdentityServerDataSeeder will be using these information for initial data seeding:

```
```json
"IdentityServer": { // Rename OpenIddict to IdentityServer
    "Clients": { // Rename Applications to Clients
        ...
    }
}
```

```

#### #### Test Project

- In **\*\*MyApplicationTestBaseModule.cs\*\*** **\*\*add\*\*** the IdentityServer related using and PreConfigurations:

```
```csharp
using Volo.Abp.IdentityServer;
```

```

and

```
```csharp
PreConfigure<AbpIdentityServerBuilderOptions>(options =>
{
    options.AddDeveloperSigningCredential = false;
});

PreConfigure<IIdentityServerBuilder>(identityServerBuilder =>
{
    identityServerBuilder.AddDeveloperSigningCredential(false,
System.Guid.NewGuid().ToString());
});
```

```

~~~

to `PreConfigureServices` to run authentication related unit tests.

UI Layer

You can follow the migrations guides from IdentityServer to OpenIddict in ****reverse order**** to update your UIs. You can also check the source-code for [\[Index.cshtml.cs\]](#) (<https://github.com/abpframework/abp-samples/blob/master/OpenId2Ids/src/OpenId2Ids.AuthServer/Pages/Index.cshtml.cs>) and [\[Index.cshtml\]](#) (<https://github.com/abpframework/abp-samples/blob/master/OpenId2Ids/src/OpenId2Ids.AuthServer/Pages/Index.cshtml>) files for ****AuthServer**** project.

- [\[Angular UI Migration\]](#) ([OpenIddict-Angular.md](#))
- [\[MVC/Razor UI Migration\]](#) ([OpenIddict-Mvc.md](#))
- [\[Blazor-Server UI Migration\]](#) ([OpenIddict-Blazor-Server.md](#))
- [\[Blazor-Wasm UI Migration\]](#) ([OpenIddict-Blazor.md](#))

Source code of samples and module

- * [\[Open source tiered & separate auth server application migrate OpenIddict to Identity Server\]](#) (<https://github.com/abpframework/abp-samples/tree/master/OpenId2Ids>)
- * [\[IdentityServer module document\]](#) (<https://docs.abp.io/en/abp/6.0/Modules/IdentityServer>)
- * [\[IdentityServer module source code\]](#) (<https://github.com/abpframework/abp/tree/release-6.0/modules/identityserver>)

16.10 OpenIddict

ABP OpenIddict Module

OpenIddict module provides an integration with the [\[OpenIddict\]](#) (<https://github.com/openiddict/openiddict-core>) which provides advanced authentication features like single sign-on, single log-out, and API access control. This module persists applications, scopes, and other OpenIddict-related objects to the database.

How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as a package and get updates easily, or you can include its source code into your solution (see `get-source` [\[CLI\]](#) ([../.CLI.md](#)) command) to develop your custom module.

The Source Code

The source code of this module can be accessed [\[here\]](#) (<https://github.com/abpframework/abp/tree/dev/modules/openiddict>). The source code is licensed by [\[MIT\]](#) (<https://choosealicense.com/licenses/mit/>), so you can freely use and customize it.

User Interface

This module implements the domain logic and database integrations but does not provide any UI. Management UI is useful if you need to add applications and scopes on the fly. In this case, you may build the management UI yourself or consider purchasing the [ABP Commercial](<https://commercial.abp.io/>) which provides the management UI for this module.

Relations to Other Modules

This module is based on the [Identity Module](Identity.md) and has an [integration package](<https://www.nuget.org/packages/Volo.Abp.Account.Web.OpenIdDict>) with the [Account Module](Account.md).

Options

OpenIdDictBuilder

`OpenIdDictBuilder` can be configured in the `PreConfigureServices` method of your OpenIdDict [module](<https://docs.abp.io/en/abp/latest/Module-Development-Basics>).

Example:

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
 PreConfigure<OpenIdDictBuilder>(builder =>
 {
 //Set options here...
 });
}
```

```

`OpenIdDictBuilder` contains various extension methods to configure the OpenIdDict services:

- `AddServer()` registers the OpenIdDict token server services in the DI container. Contains `OpenIdDictServerBuilder` configurations.
- `AddCore()` registers the OpenIdDict core services in the DI container. Contains `OpenIdDictCoreBuilder` configurations.
- `AddValidation()` registers the OpenIdDict token validation services in the DI container. Contains `OpenIdDictValidationBuilder` configurations.

OpenIdDictCoreBuilder

`OpenIdDictCoreBuilder` contains extension methods to configure the OpenIdDict core services.

Example:

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
}
```

```

```

        PreConfigure<OpenIddictCoreBuilder>(builder =>
    {
    //Set options here...
});
}
```

```

These services contain:

- Adding `ApplicationStore`, `AuthorizationStore`, `ScopeStore`, `TokenStore`.
- Replacing `ApplicationManager`, `AuthorizationManager`, `ScopeManager`, `TokenManager`.
- Replacing `ApplicationStoreResolver`, `AuthorizationStoreResolver`, `ScopeStoreResolver`, `TokenStoreResolver`.
- Setting `DefaultApplicationEntity`, `DefaultAuthorizationEntity`, `DefaultScopeEntity`, `DefaultTokenEntity`.

#### #### OpenIddictServerBuilder

`OpenIddictServerBuilder` contains extension methods to configure OpenIddict server services.

Example:

```

```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
    PreConfigure<OpenIddictServerBuilder>(builder =>
    {
    //Set options here...
});
}
```

```

These services contain:

- Registering claims, scopes.
- Setting the `Issuer` URI that is used as the base address for the endpoint URIs returned from the discovery endpoint.
- Adding development signing keys, encryption/signing keys, credentials, and certificates.
- Adding/removing event handlers.
- Enabling/disabling grant types.
- Setting authentication server endpoint URIs.

#### #### OpenIddictValidationBuilder

`OpenIddictValidationBuilder` contains extension methods to configure OpenIddict validation services.

Example:

```

```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)

```

```
{  
    PreConfigure<OpenIddictValidationBuilder>(builder =>  
    {  
        //Set options here...  
    });  
}  
...
```

16.10.1 OpenIddict Migration Guide

Migrating from IdentityServer to OpenIddict Step by Step Guide

This guide provides layer-by-layer guidance for migrating your existing application to [[OpenIddict](#)] (<https://github.com/openiddict/openiddict-core>) from IdentityServer. ABP startup templates use `OpenIddict` OpenId provider from v6.0.0 by default and `IdentityServer` projects are renamed to `AuthServer` in tiered/separated solutions. Since OpenIddict is only available with ABP v6.0, you will need to update your existing application in order to apply OpenIddict changes.

History

We are not removing Identity Server packages and we will continue to release new versions of IdentityServer-related NuGet/NPM packages. That means you won't have an issue while upgrading to v6.0 when the stable version releases. We will continue to fix bugs in our packages for a while. ABP 7.0 will be based on .NET 7. If Identity Server continues to work with .NET 7, we will also continue to ship NuGet packages for our IDS integration.

On the other hand, Identity Server ends support for the open-source Identity Server at the end of 2022. The Identity Server team has decided to move to Duende IDS and ABP will not be migrated to the commercial Duende IDS. You can see the Duende Identity Server announcement from [[this link](#)] (https://blog.duendesoftware.com/posts/20220111_fair_trade).

Commercial Template

If you are using a commercial template, please check [[Migrating from IdentityServer to OpenIddict for the Commercial Templates](#)] (<https://docs.abp.io/en/commercial/6.0/migration-guides/openiddict-step-by-step>) guide.

If you are using the microservice template, please check [[Migrating the Microservice Template from IdentityServer to OpenIddict](#)] (<https://docs.abp.io/en/commercial/6.0/migration-guides/openiddict-microservice>) guide.

OpenIddict Migration Steps

Use the `abp update` command to update your existing application. See [[Upgrading docs](#)] ([..../Upgrading.md](https://docs.abp.io/en/commercial/6.0/upgrading)) for more info. Apply required migrations by following the [[Migration Guides](#)] ([Index.md](https://docs.abp.io/en/commercial/6.0/migration-guides)) based on your application version.

Domain Shared Layer

- In **MyApplication.Domain.Shared.csproj** replace **project reference**:

```
```csharp
<PackageReference Include="Volo.Abp.IdentityServer.Domain.Shared" Version="6.0.*" />
```
```

with

```
```csharp
<PackageReference Include="Volo.Abp.OpenIddict.Domain.Shared" Version="6.0.*" />
```
```

- In ****MyApplicationDomainSharedModule.cs**** replace usings and ****module dependencies:****

```
```csharp
using Volo.Abp.IdentityServer;
...
typeof(AbpIdentityServerDomainSharedModule)
```
```

with

```
```csharp
using Volo.Abp.OpenIddict;
...
typeof(AbpOpenIddictDomainSharedModule)
```
```

Domain Layer

- In ****MyApplication.Domain.csproj**** replace ****project references****:

```
```csharp
<PackageReference Include="Volo.Abp.IdentityServer.Domain" Version="6.0.*" />
<PackageReference Include="Volo.Abp.PermissionManagement.Domain.IdentityServer"
Version="6.0.*" />
```
```

with

```
```csharp
<PackageReference Include="Volo.Abp.OpenIddict.Domain" Version="6.0.*" />
<PackageReference Include="Volo.Abp.PermissionManagement.Domain.OpenIddict"
Version="6.0.*" />
```
```

- In ****MyApplicationDomainModule.cs**** replace usings and ****module dependencies:****

```
```csharp
using Volo.Abp.IdentityServer;
using Volo.Abp.PermissionManagement.IdentityServer;
...
typeof(AbpIdentityServerDomainModule),
```

```
typeof(AbpPermissionManagementDomainIdentityServerModule),
```
```

with

```
```csharp  
using Volo.Abp.OpenIddict;
using Volo.Abp.PermissionManagement.OpenIddict;
...
typeof(AbpOpenIddictDomainModule),
typeof(AbpPermissionManagementDomainOpenIddictModule),
```
```

OpenIddictDataSeedContributor

- Create a folder named **OpenIddict** under the Domain project and copy the [\[OpenIddictDataSeedContributor.cs\]](#) (<https://github.com/abpframework/abp-samples/blob/master/Ids20OpenId/src/Ids20OpenId.Domain/OpenIddict/OpenIddictDataSeedContributor.cs>) under this folder. ****Rename**** all the `Ids20OpenId` with your project name.
- Delete **IdentityServer** folder that contains `IdentityServerDataSeedContributor.cs` which is no longer needed.

You can also create a project with the same name and copy the `OpenIddict` folder of the new project into your project.

EntityFrameworkCore Layer

If you are using MongoDB, skip this step and check the **MongoDB** layer section.

- In ****MyApplication.EntityFrameworkCore.csproj**** replace ****project reference****:

```
```csharp  
<PackageReference Include="Volo.Abp.IdentityServer.EntityFrameworkCore" Version="6.0.*"
/>
```
```

with

```
```csharp  
<PackageReference Include="Volo.Abp.OpenIddict.EntityFrameworkCore" Version="6.0.*" />
```
```

- In ****MyApplicationEntityFrameworkCoreModule.cs**** replace usings and ****module dependencies****:

```
```csharp  
using Volo.Abp.IdentityServer.EntityFrameworkCore;
...
typeof(AbpIdentityServerEntityFrameworkCoreModule),
```
```

with

```
```csharp
using Volo.Abp.OpenIddict.EntityFrameworkCore;
...
typeof(AbpOpenIddictEntityFrameworkCoreModule),
```
```

- In ****MyApplicationDbContext.cs**** replace usings and ****fluent api configurations****:

```
```csharp
using Volo.Abp.IdentityServer.EntityFrameworkCore;
...
using Volo.Abp.OpenIddict.EntityFrameworkCore;
...
protected override void OnModelCreating(ModelBuilder builder)
{
 base.OnModelCreating(builder);

 /* Include modules to your migration db context */

 ...
 builder.ConfigureIdentityServer();
}
```
```

with

```
```csharp
using Volo.Abp.OpenIddict.EntityFrameworkCore;
...
protected override void OnModelCreating(ModelBuilder builder)
{
 base.OnModelCreating(builder);

 /* Include modules to your migration db context */

 ...
 builder.ConfigureOpenIddict();
}
```
```

MongoDB Layer

If you are using EntityFrameworkCore, skip this step and check the **EntityFrameworkCore** layer section.

- In ****MyApplication.MongoDB.csproj**** replace ****project reference****:

```
```csharp
<PackageReference Include="Volo.Abp.IdentityServer.MongoDB" Version="6.0.*" />
```
```

with

```
```csharp
<PackageReference Include="Volo.Abp.OpenIddict.MongoDB" Version="6.0.*" />
```
```

- In ****MyApplicationMongoDbModule.cs**** replace usings and ****module dependencies****:

```
```csharp
using Volo.Abp.IdentityServer.MongoDB;
...
typeof(AbpIdentityServerMongoDbModule),
```
```

with

```
```csharp
using Volo.Abp.OpenIddict.MongoDB;
...
typeof(AbpOpenIddictMongoDbModule),
```
```

DbMigrator Project

- In ****MyApplication.DbMigrator.csproj** **add project reference****:

```
```csharp
<PackageReference Include="Microsoft.Extensions.Hosting" Version="6.0.1" />
```
```

for creating the host builder.

- In `appsettings.json` ****replace IdentityServer section with OpenIddict:****

```
```json
"OpenIddict": {
 "Applications": {
 "MyApplication_Web": {
 "ClientId": "MyApplication_Web",
 "ClientSecret": "1q2w3e*",
 "RootUrl": "https://localhost:44384"
 },
 "MyApplication_App": {
 "ClientId": "MyApplication_App",
 "RootUrl": "http://localhost:4200"
 },
 "MyApplication_BazorServerTiered": {
 "ClientId": "MyApplication_BazorServerTiered",
 "ClientSecret": "1q2w3e*",
 "RootUrl": "https://localhost:44346"
 },
 "MyApplication_Swagger": {
 "ClientId": "MyApplication_Swagger",
 "RootUrl": "https://localhost:44391"
 }
 }
}
```

```
 }
 }
}
```

Replace **\*\*MyApplication\*\*** with your application name.

#### #### Test Project

- In **\*\*MyApplicationTestBaseModule.cs\*\*** **\*\*remove\*\*** the IdentityServer related using and PreConfigurations:

```
```csharp
using Volo.Abp.IdentityServer;
````
```

and

```
```csharp
PreConfigure<AbpIdentityServerBuilderOptions>(options =>
{
    options.AddDeveloperSigningCredential = false;
});

PreConfigure<IIdentityServerBuilder>(identityServerBuilder =>
{
    identityServerBuilder.AddDeveloperSigningCredential(false,
System.Guid.NewGuid().ToString());
});
````
```

from `PreConfigureServices`.

#### #### UI Layer

- [Angular UI Migration] (OpenIdict-Angular.md)
- [MVC/Razor UI Migration] (OpenIdict-Mvc.md)
- [Blazor-Server UI Migration] (OpenIdict-Blazor-Server.md)
- [Blazor-Wasm UI Migration] (OpenIdict-Blazor.md)

#### ## Source code of samples and module

- \* [Open source tiered & separate auth server application migrate Identity Server to OpenIdict] (<https://github.com/abpframework/abp-samples/tree/master/Ids2OpenId>)
- \* [OpenIdict module document] (<https://docs.abp.io/en/abp/6.0/Modules/OpenIdict>)
- \* [OpenIdict module source code] (<https://github.com/abpframework/abp/tree/releases/6.0/modules/openiddict>)

#### ## See Also

- \* [ABP Version 6.0 Migration Guide] (Abp-6\_0.md)

## 16.11 Permission Management

### # Permission Management Module

This module implements the `IPermissionStore` to store and manage permissions values in a database.

➤ This document covers only the permission management module which persists permission values to a database. See the [Authorization document](./Authorization.md) to understand the authorization and permission systems.

### ## How to Install

This module comes as pre-installed (as NuGet/NPM packages). You can continue to use it as package and get updates easily, or you can include its source code into your solution (see `get-source` [CLI](./CLI.md) command) to develop your custom module.

### ### The Source Code

The source code of this module can be accessed [here](https://github.com/abpframework/abp/tree/dev/modules/permission-management). The source code is licensed with [MIT](https://choosealicense.com/licenses/mit/), so you can freely use and customize it.

### ## User Interface

#### ### Permission Management Dialog

Permission management module provides a reusable dialog to manage permissions related to an object. For example, the [Identity Module](Identity.md) uses it to manage permissions of users and roles. The following image shows Identity Module's Role Management page:

![permissions-module-open-dialog](./images/permissions-module-open-dialog.png)

When you click *\*Actions\* -> \*Permissions\** for a role, the permission management dialog is opened. An example screenshot from this dialog:

![permissions-module-dialog](./images/permissions-module-dialog.png)

In this dialog, you can grant permissions for the selected role. The tabs in the left side represents main permission groups and the right side contains the permissions defined in the selected group.

### ## IPermissionManager

`IPermissionManager` is the main service provided by this module. It is used to read and change the permission values. `IPermissionManager` is typically used by the *\*Permission Management Dialog\**. However, you can inject it if you need to set a permission value.

> If you just want to read/check permission values for the current user, use the ``IAuthorizationService`` or the `[`Authorize`]` attribute as explained in the [[Authorization document](#)](./Authorization.md).

**\*\*Example: Grant permissions to roles and users using the `IPermissionManager` service\*\***

```
```csharp
public class MyService : ITransientDependency
{
    private readonly IPermissionManager _permissionManager;

    public MyService(IPermissionManager permissionManager)
    {
        _permissionManager = permissionManager;
    }

    public async Task GrantRolePermissionDemoAsync(
        string roleName, string permission)
    {
        await _permissionManager
            .SetForRoleAsync(roleName, permission, true);
    }

    public async Task GrantUserPermissionDemoAsync(
        Guid userId, string roleName, string permission)
    {
        await _permissionManager
            .SetForUserAsync(userId, permission, true);
    }
}
````
```

## ## Permission Management Providers

Permission Management Module is extensible, just like the [[permission system](#)](./Authorization.md). You can extend it by defining permission management providers.

[[Identity Module](#)](Identity.md) defines the following permission management providers:

- \* `UserPermissionManagementProvider` : Manages user-based permissions.
- \* `RolePermissionManagementProvider` : Manages role-based permissions.

`IPermissionManager` uses these providers when you get/set permissions. You can define your own provider by implementing the `IPermissionManagementProvider` or inheriting from the `PermissionManagementProvider` base class.

**\*\*Example:\*\***

```
```csharp
public class CustomPermissionManagementProvider : PermissionManagementProvider
{
```

```

public override string Name => "Custom";

public CustomPermissionManagementProvider(
    IPermissionGrantRepository permissionGrantRepository,
    IGuidGenerator guidGenerator,
    ICurrentTenant currentTenant)
: base(
    permissionGrantRepository,
    guidGenerator,
    currentTenant)
{
}
```

```

The `PermissionManagementProvider` base class makes the default implementation (using the `IPermissionGrantRepository`) for you. You can override base methods as you need. Every provider must have a unique name, which is `Custom` in this example (keep it short since it is saved to database for each permission value record).

Once you create your provider class, you should register it using the `PermissionManagementOptions` [options class](./Options.md):

```

```csharp
Configure<PermissionManagementOptions>(options =>
{
    options.ManagementProviders.Add<CustomPermissionManagementProvider>();
});
```

```

The order of the providers are important. Providers are executed in the reverse order. That means the `CustomPermissionManagementProvider` is executed first for this example. You can insert your provider in any order in the `Providers` list.

## ## See Also

- \* [Authorization](./Authorization.md)

## 16.12 Setting Management

### # Setting Management Module

Setting Management Module implements the `ISettingStore` (see [the setting system](./Settings.md)) to store the setting values in a database and provides the `ISettingManager` to manage (change) the setting values in the database.

> Setting Management module is already installed and configured for [the startup templates](./Startup-Templates/Index.md). So, most of the times you don't need to manually add this module to your application.

### ## ISettingManager

``ISettingManager`` is used to get and set the values for the settings. Examples:

```
```csharp
using System;
using System.Threading.Tasks;
using Volo.Abp.DependencyInjection;
using Volo.Abp.SettingManagement;

namespace Demo
{
    public class MyService : ITransientDependency
    {
        private readonly ISettingManager _settingManager;

        //Inject ISettingManager service
        public MyService(ISettingManager settingManager)
        {
            _settingManager = settingManager;
        }

        public async Task FooAsync()
        {
            Guid user1Id = ...;
            Guid tenant1Id = ...;

            //Get/set a setting value for the current user or the specified user

            string layoutType1 =
                await _settingManager.GetOrNullForCurrentUserAsync("App.UI.LayoutType");
            string layoutType2 =
                await _settingManager.GetOrNullForUserAsync("App.UI.LayoutType", user1Id);

            await _settingManager.SetForCurrentUserAsync("App.UI.LayoutType", "LeftMenu");
            await _settingManager.SetForUserAsync(user1Id, "App.UI.LayoutType",
"LeftMenu");

            //Get/set a setting value for the current tenant or the specified tenant

            string layoutType3 =
                await _settingManager.GetOrNullForCurrentTenantAsync("App.UI.LayoutType");
            string layoutType4 =
                await _settingManager.GetOrNullForTenantAsync("App.UI.LayoutType",
tenant1Id);

            await _settingManager.SetForCurrentTenantAsync("App.UI.LayoutType",
"LeftMenu");
            await _settingManager.SetForTenantAsync(tenant1Id, "App.UI.LayoutType",
"LeftMenu");

            //Get/set a global and default setting value
        }
    }
}
```

```

        string layoutType5 =
            await _settingManager.GetOrNullGlobalAsync("App.UI.LayoutType");
        string layoutType6 =
            await _settingManager.GetOrNullDefaultAsync("App.UI.LayoutType");

        await _settingManager.SetGlobalAsync("App.UI.LayoutType", "TopMenu");
    }
}
```

```

So, you can get or set a setting value for different setting value providers (Default, Global, User, Tenant... etc).

> Use the `ISettingProvider` instead of the `ISettingManager` if you only need to read the setting values, because it implements caching and supports all deployment scenarios. You can use the `ISettingManager` if you are creating a setting management UI.

### ### Setting Cache

Setting values are cached using the [distributed cache](../Caching.md) system. Always use the `ISettingManager` to change the setting values which manages the cache for you.

## ## Setting Management Providers

Setting Management module is extensible, just like the [setting system](../Settings.md). You can extend it by defining setting management providers. There are 5 pre-built setting management providers registered it the following order:

- \* `DefaultValueSettingManagementProvider`: Gets the value from the default value of the setting definition. It can not set the default value since default values are hard-coded on the setting definition.
- \* `ConfigurationSettingManagementProvider`: Gets the value from the [IConfiguration service](../Configuration.md). It can not set the configuration value because it is not possible to change the configuration values on runtime.
- \* `GlobalSettingManagementProvider`: Gets or sets the global (system-wide) value for a setting.
- \* `TenantSettingManagementProvider`: Gets or sets the setting value for a tenant.
- \* `UserSettingManagementProvider`: Gets the setting value for a user.

`ISettingManager` uses the setting management providers on get/set methods. Typically, every setting management provider defines extension methods on the `ISettingManagement` service (like `SetForUserAsync` defined by the user setting management provider).

If you want to create your own provider, implement the `ISettingManagementProvider` interface or inherit from the `SettingManagementProvider` base class:

```

````csharp
public class CustomSettingProvider : SettingManagementProvider, ITransientDependency
{
    public override string Name => "Custom";
}
````
```

```
public CustomSettingProvider(ISettingManagementStore store)
 : base(store)
{
}
````
```

The `SettingManagementProvider` base class makes the default implementation (using the `ISettingManagementStore`) for you. You can override base methods as you need. Every provider must have a unique name, which is `Custom` in this example (keep it short since it is saved to database for each setting value record).

Once you create your provider class, you should register it using the `SettingManagementOptions` [options class] (./Options.md) :

```
```csharp
Configure<SettingManagementOptions>(options =>
{
 options.Providers.Add<CustomSettingProvider>();
});
````
```

The order of the providers are important. Providers are executed in the reverse order. That means the `CustomSettingProvider` is executed first for this example. You can insert your provider in any order in the `Providers` list.

See Also

* [Settings] (./Settings.md)

Setting Management UI

Setting Management module provided the email setting UI by default.

![EmailSettingUi] (./images/setting-management-email-ui.png)

➤ You can click the Send test email button to send a test email to check your email settings.

Setting it is extensible; You can add your tabs to this page for your application settings.

MVC UI

Create a setting View Component

Create `MySettingGroup` folder under the `Components` folder. Add a new view component. Name it as `MySettingGroupViewComponent` :

![MySettingGroupViewComponent] (./images/my-setting-group-view-component.png)

Open the `MySettingGroupViewComponent.cs` and change the whole content as shown below:

```
```csharp
public class MySettingGroupViewComponent : AbpViewComponent
{
 public virtual IViewComponentResult Invoke()
 {
 return View("~/Components/MySettingGroup/Default.cshtml");
 }
}
```

```

> You can also use the `InvokeAsync` method, In this example, we use the `Invoke` method.

Default.cshtml

Create a `Default.cshtml` file under the `MySettingGroup` folder.

Open the `Default.cshtml` and change the whole content as shown below:

```
```html
<div>
 <p>My setting group page</p>
</div>
```

```

BookStoreSettingPageContributor

Create a `BookStoreSettingPageContributor.cs` file under the `Settings` folder:

![BookStoreSettingPageContributor](./images/my-setting-group-page-contributor.png)

The content of the file is shown below:

```
```csharp
public class BookStoreSettingPageContributor : ISettingPageContributor
{
 public Task ConfigureAsync(SettingPageCreationContext context)
 {
 context.Groups.Add(
 new SettingPageGroup(
 "Volo.Abp.MySettingGroup",
 "MySettingGroup",
 typeof(MySettingGroupViewComponent),
 order: 1
)
);
 }

 return Task.CompletedTask;
}

public Task<bool> CheckPermissionsAsync(SettingPageCreationContext context)

```

```

 {
 // You can check the permissions here
 return Task.FromResult(true);
 }
}
```

```

Open the `BookStoreWebModule.cs` file and add the following code:

```

```csharp
Configure<SettingManagementPageOptions>(options =>
{
 options CONTRIBUTORS.Add(new BookStoreSettingPageContributor());
});
```

```

Run the Application

Navigate to `/SettingManagement` route to see the changes:

![Custom Settings Tab](./images/my-setting-group-ui.png)

Blazor UI

Create a Razor Component

Create `MySettingGroup` folder under the `Pages` folder. Add a new razor component. Name it as `MySettingGroupComponent`:

![MySettingGroupComponent](./images/my-setting-group-component.png)

Open the `MySettingGroupComponent.razor` and change the whole content as shown below:

```

```csharp
<Row>
 <p>my setting group</p>
</Row>
```

```

BookStoreSettingComponentContributor

Create a `BookStoreSettingComponentContributor.cs` file under the `Settings` folder:

![BookStoreSettingComponentContributor](./images/my-setting-group-component-contributor.png)

The content of the file is shown below:

```

```csharp
public class BookStoreSettingComponentContributor : ISettingComponentContributor
{
 public Task ConfigureAsync(SettingComponentCreationContext context)

```

```

 {
 context.Groups.Add(
 new SettingComponentGroup(
 "Volo.Abp.MySettingGroup",
 "MySettingGroup",
 typeof(MySettingGroupComponent),
 order : 1
)
);
 }

 return Task.CompletedTask;
}

public Task<bool> CheckPermissionsAsync(SettingComponentCreationContext context)
{
 // You can check the permissions here
 return Task.FromResult(true);
}
```

```

Open the `BookStoreBlazorModule.cs` file and add the following code:

```

```csharp
Configure<SettingManagementComponentOptions>(options =>
{
 options CONTRIBUTORS.Add(new BookStoreSettingComponentContributor());
});
```

```

Run the Application

Navigate to `/setting-management` route to see the changes:

![Custom Settings Tab](./images/my-setting-group-blazor.png)

Angular UI

Create a Component

Create a component with the following command:

```

```bash
yarn ng generate component my-settings
```

```

Open the `app.component.ts` and modify the file as shown below:

```

```js
import { Component } from '@angular/core';
import { SettingTabsService } from '@abp/ng.setting-management/config'; // imported
SettingTabsService
```

```

```
import { MySettingsComponent } from './my-settings/my-settings.component'; // imported
MySettingsComponent

@Component(/* component metadata */)
export class AppComponent {
  constructor(private settingTabs: SettingTabsService) // injected MySettingsComponent
  {
    // added below
    settingTabs.add([
      {
        name: 'MySettings',
        order: 1,
        requiredPolicy: 'policy key here',
        component: MySettingsComponent,
      },
    ]);
  }
}
```

Run the Application

Navigate to `/setting-management` route to see the changes:

![Custom Settings Tab](./images/custom-settings.png)

16.13 Tenant Management

Tenant Management Module

[Multi-Tenancy](./Multi-Tenancy.md) is one of the core features of ABP Framework. It provides the fundamental infrastructure to build your own SaaS (Software-as-a-Service) solution. ABP's multi-tenancy system abstracts where your tenants are stored, by providing the `ITenantStore` interface. All you need to do is to implement that interface.

****The Tenant Management module is an implementation of the the `ITenantStore` interface. It stores tenants in a database. It also provides UI to manage your tenants and their [features](./Features.md).****

> Please **refer to the [Multi-Tenancy](./Multi-Tenancy.md) documentation** to understand the multi-tenancy system of the ABP Framework. This document focuses on the Tenant Management module.

About the ABP Commercial SaaS Module

The [SaaS Module](https://commercial.abp.io/modules/Volo.SaaS) is an alternative implementation of this module with more features and possibilities. It is distributed as a part of the [ABP Commercial](https://commercial.abp.io/) subscription.

How to Install

This module comes as pre-installed (as NuGet/NPM packages) when you [[create a new solution](#)] (<https://abp.io/get-started>) with the ABP Framework. You can continue to use it as package and get updates easily, or you can include its source code into your solution (see `get-source` [[CLI](#)] ([../CLI.md](#)) command) to develop your custom module.

The Source Code

The source code of this module can be accessed [[here](#)] (<https://github.com/abpframework/abp/tree/dev/modules/tenant-management>). The source code is licensed with [[MIT](#)] (<https://choosealicense.com/licenses/mit/>), so you can freely use and customize it.

User Interface

This module adds "**Administration -> Tenant Management -> Tenants**" menu item to the main menu of the application, which opens the page shown below:

![[module-tenant-management-page](#)] ([..../images/module-tenant-management-page.png](#))

In this page, you see the all the tenants. You can create a new tenant as shown below:

![[module-tenant-management-new-tenant](#)] ([..../images/module-tenant-management-new-tenant.png](#))

In this modal;

- * ****Name****: The unique name of the tenant. If you use subdomains for your tenants (like <https://some-tenant.your-domain.com>), this will be the subdomain name.
- * ****Admin Email Address****: Email address of the admin user for this tenant.
- * ****Admin Password****: The password of the admin user for this tenant.

When you click to **Actions** button near to a tenant, you will see the actions you can take:

![[module-tenant-management-actions](#)] ([..../images/module-tenant-management-actions.png](#))

Managing the Tenant Features

The Features action opens a modal to enable/disable/set [[features](#)] ([..../Features.md](#)) for the related tenant. Here, an example modal:

![[features-modal](#)] ([..../images/features-modal.png](#))

Managing the Host Features

Manage Host features button is used to set features for the host side, if you use the features of your application also in the host side.

Distributed Events

This module defines the following ETOs (Event Transfer Objects) to allow you to subscribe to changes on the entities of the module;

- `TenantEto` is published on changes done on an `Tenant` entity.

****Example: Get notified when a new tenant has been created****

```
```
public class MyHandler :
 IDistributedEventHandler<EntityCreatedEto<TenantEto>>,
 ITransientDependency
{
 public async Task HandleEventAsync(EntityCreatedEto<TenantEto> eventData)
 {
 TenantEto tenant = eventData.Entity;
 // TODO: ...
 }
}
```

```

`TenantEto` is configured to automatically publish the events. You should configure yourself for the others. See the [Distributed Event Bus document](<https://github.com/abpframework/abp/blob/rel-7.3/docs/en/Distributed-Event-Bus.md>) to learn details of the pre-defined events.

> Subscribing to the distributed events is especially useful for distributed scenarios (like microservice architecture). If you are building a monolithic application, or listening events in the same process that runs the Tenant Management Module, then subscribing to the [local events](<https://github.com/abpframework/abp/blob/rel-7.3/docs/en/Local-Event-Bus.md>) can be more efficient and easier.

Internals

This section can be used as a reference if you want to [customize]([./Customizing-Application-Modules-Guide.md](#)) this module without changing [its source code](<https://github.com/abpframework/abp/tree/dev/modules/tenant-management>).

Domain Layer

Aggregates

* `Tenant`

Repositories

* `ITenantRepository`

Domain Services

* `TenantManager`

Application Layer

Application Services

* `TenantAppService`

Permissions

- `AbpTenantManagement.Tenants`: Tenant management.
- `AbpTenantManagement.Tenants.Create`: Creating a new tenant.
- `AbpTenantManagement.Tenants.Update`: Editing an existing tenant.
- `AbpTenantManagement.Tenants.Delete`: Deleting an existing tenant.
- `AbpTenantManagement.Tenants.ManageFeatures`: Manage features of the tenants.

Entity Framework Core Integration

* `TenantManagementDbContext` (implements `ITenantManagementDbContext`)

Database Tables:

- * `AbpTenants`
- * `AbpTenantConnectionStrings`

MongoDB Integration

* `TenantManagementMongoDbContext` (implements `ITenantManagementMongoDbContext`)

Database Collections:

- * `AbpTenants` (also includes the connection string)

Notices

ABP Framework allows to use **database per tenant** approach that allows a tenant can have a dedicated database. This module has the fundamental infrastructure to make that implementation possible (see its source code), however it doesn't implement the application layer and UI functionalities to provide it as an out of the box implementation. You can implement these features yourself, or consider to use the [ABP Commercial SaaS Module] (<https://docs.abp.io/en/commercial/latest/modules/saas>) that fully implements it and provides much more business features.

See Also

- * [Multi-Tenancy] (./Multi-Tenancy.md)
- * [ABP Commercial SaaS Module] (<https://docs.abp.io/en/commercial/latest/modules/saas>)

16.14 Virtual File Explorer

Virtual File Explorer Module

What is Virtual File Explorer Module?

Virtual File Explorer Module provided a simple UI to view all files in [virtual file system](../Virtual-File-System.md).

› Virtual File Explorer Module is not installed for [the startup templates](../Startup-Templates/Index.md). So, you need to manually add this module to your application.

Installation

1- Use ABP CLI

It is recommended to use the [ABP CLI](../CLI.md) to install the module, open the CMD window in the solution file (`.sln` directory, and run the following command:

```

```
abp add-module Volo.VirtualFileExplorer
```

```

› If you haven't done it yet, you first need to install the [ABP CLI](../CLI.md). For other installation options, see [the package description page](https://abp.io/package-detail/Volo.Abp.VirtualFileExplorer.Web).

2- Manually install

Or you can also manually install nuget package to `Acme.MyProject.Web` project:

* Install
[Volo.Abp.VirtualFileExplorer.Web](https://www.nuget.org/packages/Volo.Abp.VirtualFileExplorer.Web/) nuget package to `Acme.MyProject.Web` project.

`Install-Package Volo.Abp.VirtualFileExplorer.Web`

2.1- Adding Module Dependencies

* Open `MyProjectWebModule.cs` and add `typeof(AbpVirtualFileExplorerWebModule)` as shown below;

```
```csharp
[DependsOn(
 typeof(AbpVirtualFileExplorerWebModule),
 typeof(MyProjectApplicationModule),
 typeof(MyProjectEntityFrameworkCoreModule),
 typeof(AbpAutofacModule),
 typeof(AbpIdentityWebModule),
 typeof(AbpAccountWebModule),
 typeof(AbpAspNetCoreMvcUiBasicThemeModule)
)]
public class MyProjectWebModule : AbpModule
{
 //...
}
```

## ##### 2.2– Adding NPM Package

\* Open `package.json` and add `@abp/virtual-file-explorer": "^2.9.0` as shown below:

```
```json
{
  "version": "1.0.0",
  "name": "my-app",
  "private": true,
  "dependencies": {
    "@abp/aspnetcore.mvc.ui.theme.basic": "^2.9.0",
    "@abp/virtual-file-explorer": "^2.9.0"
  }
}
````
```

Then open the command line terminal in the `Acme.MyProject.Web` project folder and run the following command:

```
```bash
abp install-libs
````
```

That's all, Now run the application and Navigate to `/VirtualFileExplorer`. You will see virtual file explorer page:

![Virtual-File-Explorer](./images/virtual-file-explorer.png)

### ### Options

You can disabled virtual file explorer module via `AbpVirtualFileExplorerOptions` options:

```
```csharp
public override void PreConfigureServices(ServiceConfigurationContext context)
{
  PreConfigure<AbpVirtualFileExplorerOptions>(options =>
  {
    options.IsEnabled = false;
  });
}
````
```

## 16.15 Common

### 16.15.1 Database Tables

#### # Database Tables

This documentation describes all database tables and their purposes. You can read this documentation to get general knowledge of the database tables that come from each module.

## [Audit Logging Module] (Audit-Logging.md)

### [### AbpAuditLogs](#)

This table stores information about the audit logs in the application. Each record represents an audit log and tracks the actions performed in the application.

### [### AbpAuditLogActions](#)

This table stores information about the actions performed in the application, which are logged for auditing purposes.

#### [#### Foreign Keys](#)

| Table                          | Column | Description                                |
|--------------------------------|--------|--------------------------------------------|
| ---                            | ---    | ---                                        |
| [AbpAuditLogs] (#abpauditlogs) | Id     | Links each action to a specific audit log. |

### [### AbpEntityChanges](#)

This table stores information about entity changes in the application, which are logged for auditing purposes.

#### [#### Foreign Keys](#)

| Table                          | Column | Description                                       |
|--------------------------------|--------|---------------------------------------------------|
| ---                            | ---    | ---                                               |
| [AbpAuditLogs] (#abpauditlogs) | Id     | Links each entity change to a specific audit log. |

### [### AbpEntityPropertyChanges](#)

This table stores information about property changes to entities in the application, which are logged for auditing purposes.

#### [#### Foreign Keys](#)

| Table                                  | Column | Description                                             |
|----------------------------------------|--------|---------------------------------------------------------|
| ---                                    | ---    | ---                                                     |
| [AbpEntityChanges] (#abpentitychanges) | Id     | Links each property change to a specific entity change. |

## [## \[Background Jobs Module\] \(Background-Jobs.md\)](#)

### [### AbpBackgroundJobs](#)

This table stores information about the background jobs in the application and facilitates their efficient management and tracking. Each entry in the table contains details of a background job, including the job name, arguments, try count, next try time, last try time, abandoned status, and priority.

## [## \[Tenant Management Module\] \(Tenant-Management.md\)](#)

### [### AbpTenants](#)

This table stores information about the tenants. Each record represents a tenant and contains information about the tenant, such as name and other details.

#### ### AbpTenant.ConnectionStrings

This table stores information about the tenant database connection strings. When you define a connection string for a tenant, a new record will be added to this table. You can query this database to get connection strings by tenants.

#### #### Foreign Keys

| Table                     | Column | Description                                                                                                                |
|---------------------------|--------|----------------------------------------------------------------------------------------------------------------------------|
| ---                       | ---    | ---                                                                                                                        |
| [AbpTenants](#abptenants) | Id     | The `Id` column in the `AbpTenants` table is used to associate the tenant connection string with the corresponding tenant. |

#### ## Blogging Module

##### ### BlgUsers

This table stores information about the blog users. When a new identity user is created, a new record will be added to this table.

##### ### BlgBlogs

This table serves to store blog information and semantically separates the posts of each blog.

##### ### BlgPosts

This table stores information about the blog posts. You can query this table to get blog posts by blogs.

#### #### Foreign Keys

| Table                 | Column | Description                                             |
|-----------------------|--------|---------------------------------------------------------|
| ---                   | ---    | ---                                                     |
| [BlgBlogs](#blgblogs) | Id     | To associate the blog post with the corresponding blog. |

##### ### BlgComments

This table stores information about the comments made on blog posts. You can query this table to get comments by posts.

#### #### Foreign Keys

| Table                       | Column | Description                                       |
|-----------------------------|--------|---------------------------------------------------|
| ---                         | ---    | ---                                               |
| [BlgPosts](#blgposts)       | Id     | Links the comment to the corresponding blog post. |
| [BlgComments](#blgcomments) | Id     | Links the comment to the parent comment.          |

##### ### BlgTags

This table stores information about the tags. When a new tag is used, a new record will be added to this table. You can query this table to get tags by blogs.

#### ### BlgPostTags

This table is used to associate tags with blog posts in order to categorize and organize the content. You can query this table to get post tags by posts.

#### #### Foreign Keys

| Table                  | Column | Description                                        |
|------------------------|--------|----------------------------------------------------|
| [BlgTags] (#blgtags)   | Id     | Links the post tag to the corresponding tag.       |
| [BlgPosts] (#blgposts) | Id     | Links the post tag to the corresponding blog post. |

#### ## [CMS Kit Module] (Cms-Kit/Index.md)

#### ### CmsUsers

This table stores information about the CMS Kit module users. When a new identity user is created, a new record will be added to this table.

#### ### CmsBlogs

This table serves to store blog information and semantically separates the posts of each blog.

#### ### CmsBlogPosts

This table stores information about the blog posts. You can query this table to get blog posts by blogs.

#### #### Foreign Keys

| Table                  | Column | Description                                      |
|------------------------|--------|--------------------------------------------------|
| [CmsUsers] (#cmsusers) | Id     | Links the blog post to the corresponding author. |

#### ### CmsBlogFeatures

This table stores information about the blog features. You can query this table to get blog features by blogs.

#### ### CmsComments

This table is utilized by the [CMS Kit Comment system] (Cms-Kit/Comments.md) to store comments made on the blog posts. You can query this table to get comments by posts.

#### ### CmsTags

This table stores information about the tags. When a new tag is used, a new record will be added to this table. You can query this table to get tags by blogs.

#### [#### CmsEntityTags](#)

This table is utilized by the [Tag Management system] (Cms-Kit/Tags.md) to store tags and their relationship with various entities, thus enabling efficient categorization and organization of content. You can query this table to get entity tags by entities.

#### [#### CmsGlobalResources](#)

This table is a database table for the [CMS Kit Global Resources system] (Cms-Kit/Global-Resources.md), allowing dynamic addition of global styles and scripts.

#### [#### CmsMediaDescriptors](#)

This table is utilized by the CMS kit module to manage media files by using the [BlobStoring] (./Blob-Storing.md) module.

#### [#### CmsMenuItems](#)

This table is used by the [CMS Kit Menu system] (Cms-Kit/Menus.md) to manage and store information about dynamic public menus, including details such as menu item display names, URLs, and hierarchical relationships.

#### [#### CmsPages](#)

This table is utilized by the [CMS Kit Page system] (Cms-Kit/Pages.md) to store dynamic pages within the application, including information such as page URLs, titles, and content.

#### [#### CmsRatings](#)

This table is utilized by the [CMS Kit Rating system] (Cms-Kit/Ratings.md) to store ratings made on blog posts. You can query this table to get ratings by posts.

#### [#### CmsUserReactions](#)

This table is utilized by the [CMS Kit Reaction system] (Cms-Kit/Reactions.md) to store reactions made on blog posts. You can query this table to get reactions by posts.

### [## \[Docs Module\] \(Docs.md\)](#)

#### [#### DocsProjects](#)

This table stores project information to categorize documents according to different projects.

#### [#### DocsDocuments](#)

This table retrieves the document if it's not found in the cache. The documentation is being updated when the content is retrieved from the database.

#### [#### DocsDocumentContributors](#)

This table stores information about the contributors of the documents. You can query this table to get document contributors by documents.

#### #### Foreign Keys

| Table                            | Column | Description                                                   |
|----------------------------------|--------|---------------------------------------------------------------|
| [DocsDocuments] (#docsdocuments) | Id     | Links the document contributor to the corresponding document. |

#### ## [Feature Management Module] (Feature-Management.md)

#### ### AbpFeatureGroups

This table stores information about the feature groups in the application. For example, you can group all the features in the [`AbpFeatures`] (#abpfeatures) table related to the `Identity` module under the `Identity` group.

#### ### AbpFeatures

This table stores information about the features in the application. You can use the `Name` column to link each feature with its corresponding feature value in the [`AbpFeatureValues`] (#abpfeaturevalues) table, so that you can easily manage and organize the features.

#### ### AbpFeatureValues

This table stores the values of the features for different providers. You can use the `Name` column to link each feature value with its corresponding feature in the [`AbpFeatures`] (#abpfeatures) table, so that you can easily manage and organize the features.

#### ## [Identity Module] (Identity.md)

#### ### AbpUsers

This table stores information about the identity users in the application.

#### ### AbpRoles

This table stores information about the roles in the application. Roles are used to manage and control access to different parts of the application by assigning permissions and claims to roles and then assigning those roles to users. This table is important for managing and organizing the roles in the application, and for defining the access rights of the users.

#### ### AbpClaimTypes

This table stores information about the claim types used in the application. You can use the `Name`, `Regex` columns to filter the claim types by name, and regex pattern respectively, so that you can easily manage and track the claim types in the application.

### [#### AbpLinkUsers](#)

This table is useful for linking multiple user accounts across different tenants or applications to a single user, allowing them to easily switch between their accounts.

### [#### AbpUserClaims](#)

This table can manage user-based access control by allowing to assign claims to users, which describes the access rights of the individual user.

#### [##### Foreign Keys](#)

| Table                  | Column | Description                                     |
|------------------------|--------|-------------------------------------------------|
| ---                    | ---    | ---                                             |
| [AbpUsers] (#abpusers) | Id     | Links the user claim to the corresponding user. |

### [#### AbpUserLogins](#)

This table can store information about the user's external logins such as login with Facebook, Google, etc. and it can also be used to track the login history of users.

#### [##### Foreign Keys](#)

| Table                  | Column | Description                                     |
|------------------------|--------|-------------------------------------------------|
| ---                    | ---    | ---                                             |
| [AbpUsers] (#abpusers) | Id     | Links the user login to the corresponding user. |

### [#### AbpUserRoles](#)

This table can manage user-based access control by allowing to assign roles to users, which describe the access rights of the individual user.

#### [##### Foreign Keys](#)

| Table                  | Column | Description                                    |
|------------------------|--------|------------------------------------------------|
| ---                    | ---    | ---                                            |
| [AbpUsers] (#abpusers) | Id     | Links the user role to the corresponding user. |
| [AbpRoles] (#abproles) | Id     | Links the user role to the corresponding role. |

### [#### AbpUserTokens](#)

This table can store information about user's refresh tokens, access tokens and other tokens used in the application. It can also be used to invalidate or revoke user tokens.

#### [##### Foreign Keys](#)

| Table                  | Column | Description                                     |
|------------------------|--------|-------------------------------------------------|
| ---                    | ---    | ---                                             |
| [AbpUsers] (#abpusers) | Id     | Links the user token to the corresponding user. |

### [#### AbpOrganizationUnits](#)

This table is useful for creating and managing a hierarchical structure of the organization, allowing to group users and assign roles based on the organization structure. You can use the `Code`, `ParentId` columns to filter the organization units by code and parent id respectively, so that you can easily manage and track the organization units in the application.

#### #### Foreign Keys

| Table                                          | Column   | Description                                                  |
|------------------------------------------------|----------|--------------------------------------------------------------|
|                                                |          |                                                              |
| [AbpOrganizationUnits] (#abporganizationunits) | ParentId | Links the organization unit to its parent organization unit. |

#### ### AbpOrganizationUnitRoles

This table is useful for managing role-based access control at the level of organization units, allowing to assign different roles to different parts of the organization structure. You can use the `OrganizationUnitId`, `RoleId` columns to filter the roles by organization unit id and role id respectively, so that you can easily manage and track the roles assigned to organization units in the application.

#### #### Foreign Keys

| Table                                          | Column | Description                                                              |
|------------------------------------------------|--------|--------------------------------------------------------------------------|
|                                                |        |                                                                          |
| [AbpOrganizationUnits] (#abporganizationunits) | Id     | Links the organization unit role to the corresponding organization unit. |
| [AbpRoles] (#abproles)                         | Id     | Links the organization unit role to the corresponding role.              |

#### ### AbpUserOrganizationUnits

This table stores information about the organization units assigned to the users in the application. This table can manage user-organization unit relationships, and to group users based on the organization structure.

#### #### Foreign Keys

| Table                                          | Column | Description                                                              |
|------------------------------------------------|--------|--------------------------------------------------------------------------|
|                                                |        |                                                                          |
| [AbpUsers] (#abpusers)                         | Id     | Links the user organization unit to the corresponding user.              |
| [AbpOrganizationUnits] (#abporganizationunits) | Id     | Links the user organization unit to the corresponding organization unit. |

#### ### AbpRoleClaims

This table is useful for managing role-based access control by allowing to assign claims to roles, which describes the access rights of the users that belong to that role.

#### #### Foreign Keys

| Table                  | Column | Description                                     |
|------------------------|--------|-------------------------------------------------|
| ---                    | ---    | ---                                             |
| [AbpRoles] (#abproles) | Id     | Links the role claim to the corresponding role. |

#### ### AbpSecurityLogs

This table logs important operations and changes related to user accounts, allowing users to save the security logs for future reference.

#### ## [Permission Management] (Permission-Management.md)

#### ### AbpPermissionGroups

This table is important for managing and organizing the permissions in the application, by grouping them into logical categories.

#### ### AbpPermissions

This table is important for managing and controlling access to different parts of the application and for defining the granular permissions that make up the larger permissions or roles.

#### ### AbpPermissionGrants

The table stores and manage the permissions in the application and to keep track of permissions that are granted, to whom and when. Columns such as `Name`, `ProviderName`, `ProviderKey`, `TenantId` can be used to filter the granted permissions by name, provider name, provider key, and tenant id respectively, so that you can easily manage and track the granted permissions in the application.

#### ## [Setting Management] (Setting-Management.md)

#### ### AbpSettings

This table stores key-value pairs of settings for the application, and it allows dynamic configuration of the application without the need for recompilation.

#### ## [OpenIddict] (OpenIddict.md)

#### ### OpenIddictApplications

This table can store information about the OpenID Connect applications, including the client id, client secret, redirect URI, and other relevant information. It can also be used to authenticate and authorize clients using OpenID Connect protocol.

#### ### OpenIddictAuthorizations

This table stores the OpenID Connect authorization data in the application. It can also be used to manage and validate the authorization grants issued to clients and users.

#### #### Foreign Keys

| Table                                              | Column | Description                                               |
|----------------------------------------------------|--------|-----------------------------------------------------------|
| ---                                                | ---    | ---                                                       |
| [OpenIddictApplications] (#openiddictapplications) | Id     | Links the authorization to the corresponding application. |
|                                                    |        |                                                           |

### ### OpenIddictTokens

This table can store information about the OpenID Connect tokens, including the token payload, expiration, type, and other relevant information. It can also be used to manage and validate the tokens issued to clients and users, such as access tokens and refresh tokens, and to control access to protected resources.

### #### Foreign Keys

| Table                                                  | Column | Description                                         |
|--------------------------------------------------------|--------|-----------------------------------------------------|
| ---                                                    | ---    | ---                                                 |
| [OpenIddictApplications] (#openiddictapplications)     | Id     | Links the token to the corresponding application.   |
| [OpenIddictAuthorizations] (#openiddictauthorizations) | Id     | Links the token to the corresponding authorization. |
|                                                        |        |                                                     |

### ### OpenIddictScopes

This table can store information about the OpenID Connect scopes, including the name and description of the scope. It can also be used to define the permissions or access rights associated with the scopes, which are then used to control access to protected resources.

## ## [IdentityServer] (IdentityServer.md)

### ### IdentityServerApiResources

This table can store information about the API resources, including the resource name, display name, description, and other relevant information. It can also be used to define the scopes, claims, and properties associated with the API resources, which are then used to control access to protected resources.

### ### IdentityServerIdentityResources

This table can store information about the identity resources, including the name, display name, description, and enabled status.

### ### IdentityServerClients

This table can store information about the clients, including the client id, client name, client URI and other relevant information. It can also be used to define the scopes, claims, and properties associated with the clients, which are then used to control access to protected resources.

### ### IdentityServerApiScopes

This table can store information about the API scopes, including the scope name, display name, description, and other relevant information. It can also be used to define the claims and properties associated with the API scopes, which are then used to control access to protected resources.

#### ### IdentityServerApiResourceClaims

This table can store information about the claims of an API resource, including the claim type and API resource id.

##### #### Foreign Keys

| Table                                                     | Column | Description                                        |
|-----------------------------------------------------------|--------|----------------------------------------------------|
| ---                                                       | ---    | ---                                                |
| [IdentityServerApiResources](#identityserverapiresources) | Id     | Links the claim to the corresponding API resource. |

#### ### IdentityServerIdentityResourceClaims

This table can store information about the claims of an identity resource, including the claim type and identity resource id.

##### #### Foreign Keys

| Table                                                               | Column | Description                                             |
|---------------------------------------------------------------------|--------|---------------------------------------------------------|
| ---                                                                 | ---    | ---                                                     |
| [IdentityServerIdentityResources](#identityserveridentityresources) | Id     | Links the claim to the corresponding identity resource. |

#### ### IdentityServerClientClaims

This table can store information about the claims of a client, including the claim type, claim value and client id.

##### #### Foreign Keys

| Table                                           | Column | Description                                  |
|-------------------------------------------------|--------|----------------------------------------------|
| ---                                             | ---    | ---                                          |
| [IdentityServerClients](#identityserverclients) | Id     | Links the claim to the corresponding client. |

#### ### IdentityServerApiScopeClaims

This table can store information about the claims of an API scope, including the claim type and API scope id.

##### #### Foreign Keys

| Table                                               | Column | Description                                     |
|-----------------------------------------------------|--------|-------------------------------------------------|
| ---                                                 | ---    | ---                                             |
| [IdentityServerApiScopes](#identityserverapiscopes) | Id     | Links the claim to the corresponding API scope. |

### [### IdentityServerApiResourceProperties](#)

This table can store information about properties, including the property key and value, and the associated API resource. These properties can store additional metadata or configuration information related to the API resources.

#### [#### Foreign Keys](#)

| Table                                                     | Column | Description                                           |
|-----------------------------------------------------------|--------|-------------------------------------------------------|
|                                                           |        |                                                       |
| [IdentityServerApiResources](#identityserverapiresources) | Id     | Links the property to the corresponding API resource. |

### [### IdentityServerIdentityResourceProperties](#)

This table can store information about properties, including the property key and value, and the associated identity resource. These properties can store additional metadata or configuration information related to the identity resources.

#### [#### Foreign Keys](#)

| Table                                                               | Column | Description                                                |
|---------------------------------------------------------------------|--------|------------------------------------------------------------|
|                                                                     |        |                                                            |
| [IdentityServerIdentityResources](#identityserveridentityresources) | Id     | Links the property to the corresponding identity resource. |

### [### IdentityServerClientProperties](#)

This table can store information about the properties of a client, including the key, value and client id. These properties can store additional metadata or configuration information related to the clients.

#### [#### Foreign Keys](#)

| Table                                           | Column | Description                                     |
|-------------------------------------------------|--------|-------------------------------------------------|
|                                                 |        |                                                 |
| [IdentityServerClients](#identityserverclients) | Id     | Links the property to the corresponding client. |

### [### IdentityServerApiScopeProperties](#)

This table can store information about the properties of an API scope, including the key, value and API scope id. These properties can store additional metadata or configuration information related to the API scopes.

#### [#### Foreign Keys](#)

| Table                                               | Column | Description                                        |
|-----------------------------------------------------|--------|----------------------------------------------------|
|                                                     |        |                                                    |
| [IdentityServerApiScopes](#identityserverapiscopes) | Id     | Links the property to the corresponding API scope. |

### [### IdentityServerApiResourceScopes](#)

This table can store information about the scopes of an API resource, including the scope name and API resource id.

#### [#### Foreign Keys](#)

| Table                                                     | Column | Description                                        |
|-----------------------------------------------------------|--------|----------------------------------------------------|
|                                                           |        |                                                    |
| [IdentityServerApiResources](#identityserverapiresources) | Id     | Links the scope to the corresponding API resource. |

### [### IdentityServerClientScopes](#)

This table can store information about the scopes of a client, including the scope and client id.

#### [#### Foreign Keys](#)

| Table                                           | Column | Description                                  |
|-------------------------------------------------|--------|----------------------------------------------|
|                                                 |        |                                              |
| [IdentityServerClients](#identityserverclients) | Id     | Links the scope to the corresponding client. |

### [### IdentityServerApiResourceSecrets](#)

This table can store information about the secrets of an API resource, including the secret value, expiration date, and API resource id.

#### [#### Foreign Keys](#)

| Table                                                     | Column | Description                                         |
|-----------------------------------------------------------|--------|-----------------------------------------------------|
|                                                           |        |                                                     |
| [IdentityServerApiResources](#identityserverapiresources) | Id     | Links the secret to the corresponding API resource. |

### [### IdentityServerClientSecrets](#)

This table can store information about the secrets of a client, including the secret value, expiration date, and client id.

#### [#### Foreign Keys](#)

| Table                                           | Column | Description                                   |
|-------------------------------------------------|--------|-----------------------------------------------|
|                                                 |        |                                               |
| [IdentityServerClients](#identityserverclients) | Id     | Links the secret to the corresponding client. |

### [### IdentityServerClientCorsOrigins](#)

This table can store information about the CORS origins of a client, including the origin and client id. It can also be used to manage and validate the CORS origins of a client.

#### #### Foreign Keys

| Table                                           | Column | Description                                        |
|-------------------------------------------------|--------|----------------------------------------------------|
| ---                                             | ---    | ---                                                |
| [IdentityServerClients](#identityserverclients) | Id     | Links the CORS origin to the corresponding client. |

#### ### IdentityServerClientGrantTypes

This table can store information about the grant types of a client, including the grant type and client id.

#### #### Foreign Keys

| Table                                           | Column | Description                                       |
|-------------------------------------------------|--------|---------------------------------------------------|
| ---                                             | ---    | ---                                               |
| [IdentityServerClients](#identityserverclients) | Id     | Links the grant type to the corresponding client. |

#### ### IdentityServerClientIdPRestrictions

This table can store information about the identity provider restrictions of a client, including the identity provider and client id.

#### #### Foreign Keys

| Table                                           | Column | Description                                                          |
|-------------------------------------------------|--------|----------------------------------------------------------------------|
| ---                                             | ---    | ---                                                                  |
| [IdentityServerClients](#identityserverclients) | Id     | Links the identity provider restriction to the corresponding client. |

#### ### IdentityServerClientPostLogoutRedirectUris

This table can store information about the post logout redirect URIs of a client, including the post logout redirect URI and client id.

#### #### Foreign Keys

| Table                                           | Column | Description                                                     |
|-------------------------------------------------|--------|-----------------------------------------------------------------|
| ---                                             | ---    | ---                                                             |
| [IdentityServerClients](#identityserverclients) | Id     | Links the post logout redirect URI to the corresponding client. |

#### ### IdentityServerClientRedirectUris

This table can store information about the redirect URIs of a client, including the redirect URI and client id.

#### #### Foreign Keys

| Table                                           | Column | Description                                         |
|-------------------------------------------------|--------|-----------------------------------------------------|
| ---                                             | ---    | ---                                                 |
| [IdentityServerClients](#identityserverclients) | Id     | Links the redirect URI to the corresponding client. |

### ### IdentityServerDeviceFlowCodes

This table can store information about the device flow codes, including the user code, device code, subject id, client id, creation time, expiration, data and session id.

### ### IdentityServerPersistedGrants

This table can store information about the persisted grants, including the key, type, subject id, client id, creation time, expiration, and data.

## ## Others

### ### AbpBlobContainers

This table is important for providing a better user experience by allowing the application to support multiple containers and providing BLOB-specific features.

### ### AbpBlobs

This table stores the binary data of BLOBS (binary large objects) in the application. Each BLOB is related to a container in the [AbpBlobContainers](#abpblobcontainers) table, where the container name, tenant id and other properties of the container can be found.

## #### Foreign Keys

| Table                                   | Column | Description                                    |
|-----------------------------------------|--------|------------------------------------------------|
| ---                                     | ---    | ---                                            |
| [AbpBlobContainers](#abpblobcontainers) | Id     | Links the BLOB to the corresponding container. |

### ### AbpLocalizationResources

This table stores the localization resources for the application. This table is important for providing a better user experience by allowing the application to support multiple resources and providing localized text and other localization-specific features.

### ### AbpLocalizationTexts

The table contains the resource name, culture name, and a json encoded value which holds the key-value pair of localization text. It allows for efficient storage and management of localization texts and allows for easy update or addition of new translations for specific resources and cultures.

# 17 Samples

## 17.1 All Samples

### # Sample Applications

Here, a list of official samples built with the ABP Framework. Most of these samples are located under the [\[abpframework/abp-samples\]](#) (<https://github.com/abpframework/abp-samples>) GitHub repository.

#### ## eShopOnAbp

Reference microservice solution built with the ABP Framework and .NET.

- \* [\[Source code\]](#) (<https://github.com/abpframework/eShopOnAbp>)

#### ## EventHub

This is a reference application built with the ABP Framework. It implements the Domain Driven Design with multiple application layers.

- \* [\[Source code\]](#) (<https://github.com/abpframework/eventhub>)

#### ## Book Store

A simple CRUD application to show basic principles of developing an application with the ABP Framework. The same sample was implemented with different technologies:

- \* **\*\*Book Store: Razor Pages UI & Entity Framework Core\*\***
  - \* [\[Tutorial\]](#) (<https://docs.abp.io/en/abp/latest/Tutorials/Part-1?UI=MVC&DB=EF>)
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/BookStore-Mvc-EfCore>)
- \* **\*\*Book Store: Blazor UI & Entity Framework Core\*\***
  - \* [\[Tutorial\]](#) (<https://docs.abp.io/en/abp/latest/Tutorials/Part-1?UI=Blazor&DB=EF>)
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/BookStore-Blazor-EfCore>)
- \* **\*\*Book Store: Angular UI & MongoDB\*\***
  - \* [\[Tutorial\]](#) (<https://docs.abp.io/en/abp/latest/Tutorials/Part-1?UI=NG&DB=Mongo>)
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/BookStore-Angular-MongoDb>)
- \* **\*\*Book Store: Modular application (Razor Pages UI & EF Core)\*\***
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/BookStore-Modular>)

While there is no Razor Pages & MongoDB combination, you can check both documents to understand it since DB & UI selection don't effect each other.

#### ## Other Samples

- \* **\*\*Event Organizer\*\***: A sample application to create events (meetups) and allow others to register the events. Developed using EF Core and Blazor UI.

- \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/EventOrganizer) (<https://github.com/abpframework/abp-samples/tree/master/EventOrganizer>)
  - \* [\[Article\]](https://community.abp.io/articles/creating-an-event-organizer-application-with-the-blazor-ui-wbe0sf2z) (<https://community.abp.io/articles/creating-an-event-organizer-application-with-the-blazor-ui-wbe0sf2z>)
- \* **Entity Framework Migrations**: A solution to demonstrate how to split your application into multiple databases each database contains different modules.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/EfCoreMigrationDemo) (<https://github.com/abpframework/abp-samples/tree/master/EfCoreMigrationDemo>)
    - \* [\[EF Core database migrations document\]](https://github.com/abpframework/abp-samples/tree/master/Entity-Framework-Core-Migrations.md) ([../Entity-Framework-Core-Migrations.md](https://github.com/abpframework/abp-samples/tree/master/Entity-Framework-Core-Migrations.md))
- \* **SignalR Demo**: A simple chat application that allows to send and receive messages among authenticated users.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/SignalRDemo) (<https://github.com/abpframework/abp-samples/tree/master/SignalRDemo>)
    - \* [\[SignalR Integration document\]](https://github.com/abpframework/abp-samples/tree/master/SignalRIntegration.md) ([../SignalR-Integration.md](https://github.com/abpframework/abp-samples/tree/master/SignalRIntegration.md))
- \* **Real Time Messaging In A Distributed Architecture**: (using SingalR & RabbitMQ)
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/SignalRTieredDemo) (<https://github.com/abpframework/abp-samples/tree/master/SignalRTieredDemo>)
    - \* [\[Article\]](https://community.abp.io/articles/real-time-messaging-in-a-distributed-architecture-using-abp-framework-singalr-rabbitmq-daf47e17) (<https://community.abp.io/articles/real-time-messaging-in-a-distributed-architecture-using-abp-framework-singalr-rabbitmq-daf47e17>)
- \* **Dashboard Demo**: A simple application to show how to use the widget system for the ASP.NET Core MVC UI.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/DashboardDemo) (<https://github.com/abpframework/abp-samples/tree/master/DashboardDemo>)
    - \* [\[Widget documentation\]](https://github.com/abpframework/abp-samples/tree/master/UI/AspNetCore/Widgets.md) ([../UI/AspNetCore/Widgets.md](https://github.com/abpframework/abp-samples/tree/master/UI/AspNetCore/Widgets.md))
- \* **RabbitMQ Event Bus Demo**: A solution consists of two applications communicating to each other via distributed events with RabbitMQ integration.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/RabbitMqEventBus) (<https://github.com/abpframework/abp-samples/tree/master/RabbitMqEventBus>)
    - \* [\[Distributed event bus document\]](https://github.com/abpframework/abp-samples/tree/master/DistributedEventBus.md) ([../Distributed-Event-Bus.md](https://github.com/abpframework/abp-samples/tree/master/DistributedEventBus.md))
    - \* [\[RabbitMQ distributed event bus integration document\]](https://github.com/abpframework/abp-samples/tree/master/DistributedEventBus-RabbitMQ-Integration.md) ([../Distributed-Event-Bus-RabbitMQ-Integration.md](https://github.com/abpframework/abp-samples/tree/master/DistributedEventBus-RabbitMQ-Integration.md))
- \* **Text Templates Demo**: Shows different use cases of the text templating system.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/TextTemplateDemo) (<https://github.com/abpframework/abp-samples/tree/master/TextTemplateDemo>)
    - \* [\[Text templating documentation\]](https://github.com/abpframework/abp-samples/tree/master/TextTemplating.md) ([../Text-Templating.md](https://github.com/abpframework/abp-samples/tree/master/TextTemplating.md))
- \* **Stored Procedure Demo**: Demonstrates how to use stored procedures, database views and functions with best practices.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/StoredProcedureDemo) (<https://github.com/abpframework/abp-samples/tree/master/StoredProcedureDemo>)
- \* **Passwordless Authentication**: Shows how to add a custom token provider to authenticate a user with a link, instead of entering a password.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/PasswordlessAuthentication) (<https://github.com/abpframework/abp-samples/tree/master/PasswordlessAuthentication>)
    - \* [\[Article\]](https://community.abp.io/articles/implementing-passwordless-authentication-with-asp.net-core-identity-c2518koj) (<https://community.abp.io/articles/implementing-passwordless-authentication-with-asp.net-core-identity-c2518koj>)
- \* **Authentication Customization**: A solution to show how to customize the authentication for ASP.NET Core MVC / Razor Pages applications.
  - \* [\[Source code\]](https://github.com/abpframework/abp-samples/tree/master/AuthenticationCustomization) (<https://github.com/abpframework/abp-samples/tree/master/AuthenticationCustomization>)
- \* Related articles:
  - \* [\[Azure Active Directory Authentication\]](https://community.abp.io/articles/how-to-use-the-azure-active-directory-authentication-for-mvc-razor-page-applications-4603b9cf) (<https://community.abp.io/articles/how-to-use-the-azure-active-directory-authentication-for-mvc-razor-page-applications-4603b9cf>)
  - \* [\[Customize the Login Page\]](https://community.abp.io/articles/how-to-customize-the-login-page-for-mvc-razor-page-applications-9a40f3cd) (<https://community.abp.io/articles/how-to-customize-the-login-page-for-mvc-razor-page-applications-9a40f3cd>)

- \* [\[Customize the SignIn Manager\]](#) (<https://community.abp.io/articles/how-to-customize-the-signin-manager-3e858753>)
- \* **\*\*GRPC Demo\*\*:** Shows how to add a gRPC service to an ABP Framework based web application and consume it from a console application.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/GrpcDemo>)
- \* **\*\*Telerik Blazor Integration\*\*:** Shows how to install and use Telerik Blazor components with the ABP Framework.
  - \* [\[Article\]](#) (<https://community.abp.io/articles/how-to-integrate-the-telerik-blazor-components-to-the-abp-blazor-ui-q8g31abb>)
- \* **\*\*Angular Material Integration\*\*:** Implemented the web application tutorial using the Angular Material library.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/AcmeBookStoreAngularMaterial>)
  - \* [\[Article\]](#) (<https://community.abp.io/articles/using-angular-material-components-with-the-abp-framework-af8ft6t9>)
- \* **\*\*DevExtreme Angular Component Integration\*\*:** How to install and use DevExtreme components in the ABP Framework Angular UI.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/DevExtreme-Angular>)
  - \* [\[Article\]](#) (<https://community.abp.io/articles/using-devextreme-angular-components-with-the-abp-framework-x5nyvjj3i>)
- \* **\*\*DevExtreme MVC / Razor Pages Component Integration\*\*:** How to install and use DevExtreme components in the ABP Framework MVC / Razor Pages UI.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/DevExtreme-Mvc>)
  - \* [\[Article\]](#) (<https://community.abp.io/articles/using-devextreme-components-with-the-abp-framework-zb8z7yqv>)
- \* **\*\*Syncfusion Blazor Integration\*\*:** Shows how to install and integrate Syncfusion UI with ABP Framework Blazor UI.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/SyncfusionSample>)
  - \* [\[Article\]](#) (<https://community.abp.io/articles/using-syncfusion-components-with-the-abp-framework-5ccvi8kc>)
- \* **\*\*Empty ASP.NET Core Application\*\*:** The most basic ASP.NET Core application with the ABP Framework installed.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/BasicAspNetCoreApplication>)
  - \* [\[Documentation\]](#) ([../Getting-Started-AspNetCore-Application.md](#))
- \* **\*\*Using Elsa Workflow with ABP Framework\*\*:** Shows how to use the Elsa Core workflow library within an ABP-based application.
  - \* [\[Source code\]](#) (<https://github.com/abpframework/abp-samples/tree/master/ElsaDemo>)
  - \* [\[Article\]](#) (<https://community.abp.io/articles/using-elsa-workflow-with-the-abp-framework-773siq19>)

## 17.2 eShopOnAbp

["https://github.com/abpframework/eShopOnAbp"](https://github.com/abpframework/eShopOnAbp)

## 17.3 EventHub

"<https://github.com/abpframework/eventhub>"

## 17.4 Microservice Demo (legacy)

### # Microservice Demo Solution

➤ This solution is no longer maintained. See [[the eShopOnAbp project](#)](<https://github.com/abpframework/eShopOnAbp>) for the replacement solution.

*\*"Microservices are a software development technique—a variant of the **\*\*service-oriented architecture\*\*** (SOA) architectural style that structures an application as a collection of **\*\*loosely coupled services\*\***. In a microservices architecture, services are **\*\*fine-grained\*\*** and the protocols are **\*\*lightweight\*\***. The benefit of decomposing an application into different smaller services is that it improves **\*\*modularity\*\***. This makes the application easier to understand, develop, test, and become more resilient to architecture erosion. It **\*\*parallelizes development\*\*** by enabling small autonomous teams to **\*\*develop, deploy and scale\*\*** their respective services independently. It also allows the architecture of an individual service to emerge through **\*\*continuous refactoring\*\***. Microservices-based architectures enable **\*\*continuous delivery and deployment\*\***.\**

— [[Wikipedia](#)](<https://en.wikipedia.org/wiki/Microservices>)

### ## Introduction

One of the major goals of the ABP framework is to provide a [[convenient infrastructure to create microservice solutions](#)]([./Microservice-Architecture.md](#)).

This sample aims to demonstrate a simple yet complete microservice solution;

- \* Has multiple, independent, self-deployable **\*\*microservices\*\***.
- \* Multiple **\*\*web applications\*\***, each uses a different API gateway.
- \* Has multiple **\*\*gateways\*\*** / BFFs (Backend for Frontends) developed using the [[Ocelot](#)](<https://github.com/ThreeMammals/Ocelot>) library.
- \* Has an **\*\*authentication service\*\*** developed using the [[IdentityServer](#)](<https://identityserver.io/>) framework. It's also a SSO (Single Sign On) application with necessary UIs.
- \* Has **\*\*multiple databases\*\***. Some microservices has their own database while some services/applications shares a database (to demonstrate different use cases).
- \* Has different types of databases: **\*\*SQL Server\*\*** (with **\*\*Entity Framework Core\*\*** ORM) and **\*\*MongoDB\*\***.
- \* Has a **\*\*console application\*\*** to show the simplest way of using a service by authenticating.
- \* Uses [[Redis](#)](<https://redis.io/>) for **\*\*distributed caching\*\***.
- \* Uses [[RabbitMQ](#)](<https://www.rabbitmq.com/>) for service-to-service **\*\*messaging\*\***.
- \* Uses [[Docker](#)](<https://www.docker.com/>) & [[Kubernetes](#)](<https://kubernetes.io/>) to **\*\*deploy\*\*** & run all services and applications.
- \* Uses [[Elasticsearch](#)](<https://www.elastic.co/products/elasticsearch>) & [[Kibana](#)](<https://www.elastic.co/products/kibana>) to store and visualize the logs (written using [[Serilog](#)](<https://serilog.net/>)).

The diagram below shows the system:

![microservice-sample-diagram-2](./images/microservice-sample-diagram-3.png)

#### ### Source Code

You can get the source code from [the GitHub repository](<https://github.com/abpframework/abp-samples/tree/master/MicroserviceDemo>).

#### ## Running the Solution

##### ### Pre Requirements

To be able to run the solution from source code, following tools should be installed and running on your computer:

- \* [SQL Server] (<https://www.microsoft.com/en-us/sql-server/sql-server-downloads>) 2015+ (can be [express edition] (<https://www.microsoft.com/en-us/sql-server/sql-server-editions-express>))
- \* [Redis] (<https://redis.io/download>) 5.0+
- \* [RabbitMQ] (<https://www.rabbitmq.com/install-windows.html>) 3.7.11+
- \* [MongoDB] (<https://www.mongodb.com/download-center>) 4.0+
- \* [ElasticSearch] (<https://www.elastic.co/downloads/elasticsearch>) 6.6+
- \* [Kibana] (<https://www.elastic.co/downloads/kibana>) 6.6+ (optional, recommended to show logs)

##### ### Running Infrastructure

- \* Docker-compose is used to run the pre requirements with ease as default. If you don't have it, you can download and start using [Docker for Windows] (<https://docs.docker.com/docker-for-windows/>) from [here] (<https://docs.docker.com/docker-for-windows/install/>) on windows environment.
- \* Run the command `docker-compose -f docker-compose.infrastructure.yml -f docker-compose.override.yml up -d` at `MicroserviceDemo` directory or run the powershell script `\_\_Run\_Infrastructure.ps1` located at `MicroserviceDemo/\_run` directory.
- \* If you don't want to use docker for pre required services and install them on your local development, you need to update `appsettings.json` files of the projects in the MicroserviceDemo solution accordingly.

##### ### Open & Build the Visual Studio Solution

- \* Open the `samples\MicroserviceDemo\MicroserviceDemo.sln` in Visual Studio 2017 (15.9.0+).
- \* Run `dotnet restore` from the command line inside the `samples\MicroserviceDemo` folder.
- \* Build the solution in Visual Studio.

##### ### Create Databases

MongoDB database is created dynamically, however you need to create database schemas for SQL server databases. The solution is configured to use Entity Core Code First migrations, so you can easily create databases.

There are two SQL server databases in this solution.

#### #### MsDemo\_Identity Database

- \* Right click to the `AuthServer.Host` project and click to the `Set as startup project`.
- \* Open the \*\*Package Manager Console\*\* (Tools → Nuget Package Manager → Package Manager Console)
- \* Select `AuthServer.Host` as the \*\*Default project\*\*.
- \* Run `Update-Database` command.

![microservice-sample-update-database-authserver] (./images/microservice-sample-update-database-authserver.png)

#### #### MsDemo\_ProductManagement

- Right click to the `ProductService.Host` project and click to the `Set as startup project`.
- Open the \*\*Package Manager Console\*\* (Tools → Nuget Package Manager → Package Manager Console)
- Select `ProductService.Host` as the \*\*Default project\*\*.
- Run `Update-Database` command.

![microservice-sample-update-database-products] (./images/microservice-sample-update-database-products.png)

#### ### Run Projects

Run the projects with the following order (right click to each project, set as startup project an press Ctrl+F5 to run without debug) :

- \* AuthServer.Host
- \* IdentityService.Host
- \* TenantManagementService.Host
- \* BloggingService.Host
- \* ProductService.Host
- \* InternalGateway.Host
- \* BackendAdminAppGateway.Host
- \* PublicWebSiteGateway.Host
- \* BackendAdminApp.Host
- \* PublicWebSite.Host

When you run projects, they will add some initial demo data to their databases.

#### ## A Brief Overview of the Solution

The Visual Studio solution consists of multiple projects each have different roles in the system:

![microservice-sample-solution] (./images/microservice-sample-solution-2.png)

#### ### Applications

These are the actual applications those have user interfaces to interact to the users and use the system.

- **AuthServer.Host**: Host the IdentityServer4 to provide an authentication service to other services and applications. It is a single-sign server and contains the login page.
- **BackendAdminApp.Host**: This is a backend admin application that host UI for Identity and Product management modules.
- **PublicWebSite.Host**: As public web site that contains a simple product list page and blog module UI.
- **ConsoleClientDemo**: A simple console application to demonstrate the usage of services from a C# application.

### ### Gateways / BFFs (Backend for Frontend)

Gateways are used to provide a single entry point to the applications. It can also used for rate limiting, load balancing... etc. Used the [Ocelot](<https://github.com/ThreeMammals/Ocelot>) library.

- \* **BackendAdminAppGateway.Host**: Used by the BackendAdminApp.Host application as backend.
- \* **PublicWebSiteGateway.Host**: Used by the PublicWebSite.Host application as backend.
- \* **InternalGateway.Host**: Used for inter-service communication (the communication between microservices).

### ### Microservices

Microservices have no UI, but exposes some REST APIs.

- **IdentityService.Host**: Hosts the ABP Identity module which is used to manage users & roles. It has no additional service, but only hosts the Identity module's API.
- **TenantManagementService.Host**: Hosts the ABP Tenant Management module which is used to manage roles. It has no additional service, but only hosts the Tenant Management module's API.
- **BloggingService.Host**: Hosts the ABP Blogging module which is used to manage blog & posts (a typical blog application). It has no additional service, but only hosts the Blogging module's API.
- **ProductService.Host**: Hosts the Product module (that is inside the solution) which is used to manage products. It also contains the EF Core migrations to create/update the Product Management database schema.

### ### Modules

- \* **Product**: A layered module that is developed with the [module development best practices]([./Best-Practices/Index.md](#)). It can be embedded into a monolithic application or can be hosted as a microservice by separately deploying API and UI (as done in this demo solution).

### ### Databases

This solution is using multiple databases:

- \* **MsDemo\_Identity**: An SQL database. Used **SQL Server** by default, but can be any DBMS supported by the EF Core. Shared by AuthServer, IdentityService and the TenantManagementService. Also audit logs, permissions and settings are stored in this database (while they could have their own databases, shared the same database to keep it simple).
- \* **MsDemo\_ProductManagement**: An SQL database. Again, used **SQL Server** by default, but can be any DBMS supported by the EF Core. Used by the ProductService as a dedicated database.
- \* **MsDemo\_Blogging**: A **MongoDB** database. Used by the BloggingService.
- \* **Elasticsearch**: Used to write logs over Serilog.

## ## Applications

### ### Authentication Server (AuthServer.Host)

This project is used by all other services and applications for authentication & single sign on. Mainly, uses **IdentityServer4** to provide these services. It uses some of the [pre-build ABP modules](./Modules/Index) like *\*Identity\**, *\*Audit Logging\** and *\*Permission Management\**.

### #### Database & EF Core Configuration

This application uses a SQL database (named it as **MsDemo\_Identity**) and maintains its schema via **Entity Framework Core migrations**.

It has a DbContext named **AuthServerDbContext** and defined as shown below:

```
```csharp
public class AuthServerDbContext : AbpDbContext<AuthServerDbContext>
{
    public AuthServerDbContext(DbContextOptions<AuthServerDbContext> options)
        : base(options)
    {

    }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        base.OnModelCreating(modelBuilder);

        modelBuilder.ConfigureIdentity();
        modelBuilder.ConfigureIdentityServer();
        modelBuilder.ConfigureAuditLogging();
        modelBuilder.ConfigurePermissionManagement();
        modelBuilder.ConfigureSettingManagement();
    }
}
```

```

In the **OnModelCreating**, you see **ConfigureX()** method calls. A module with a database schema generally declares such an extension method to configure EF Core mappings for its own entities. This is a flexible approach where you can arrange your databases and

modules inside them; You can use a different database for each module, or combine some of them in a shared database. In the AuthServer project, we decided to combine multiple module schemas in a single EF Core DbContext, in a single physical database. These modules are Identity, IdentityServer, AuditLogging, PermissionManagement and SettingManagement modules.

Notice that this DbContext is only for database migrations. All modules have their own `DbContext` classes those are used in the runtime by the modules.

#### #### User Interface

AuthServer has a simple home page that shows the current user info if the current user has logged in:

![microservice-sample-authserver-home](../images/microservice-sample-authserver-home.png)

It also provides Login & Register pages:

![microservice-sample-authserver-login](../images/microservice-sample-authserver-login.png)

These pages are not included in the project itself. Instead, AuthServer project uses the prebuilt ABP [account module](<https://github.com/abpframework/abp/tree/master/modules/account>) with IdentityServer extension. That means it can also act as an OpenId Connect server with necessary UI and logic.

#### #### Dependencies

- \* \*\*RabbitMQ\*\* for messaging to other services.
- \* \*\*Redis\*\* for distributed/shared caching.
- \* \*\*Elasticsearch\*\* for storing logs.

#### ### Backend Admin Application (BackendAdminApp.Host)

This is a web application that is used to manage users, roles, permissions and products in the system.

#### #### Authentication

BackendAdminApp redirects to the AuthServer for authentication. Once the user enters a correct username & password, the page is redirected to the backend application again. Authentication configuration is setup in the `BackendAdminAppHostModule` class:

```
```csharp
context.Services.AddAuthentication(options =>
{
    options.DefaultScheme = "Cookies";
    options.DefaultChallengeScheme = "oidc";
})
.AddCookie("Cookies", options =>
{
```

```

        options.Cookie.Expiration = TimeSpan.FromDays(365);
        options.ExpireTimeSpan = TimeSpan.FromDays(365);
    })
    .AddOpenIdConnect("oidc", options =>
{
    options.Authority = configuration["AuthServer:Authority"];
    options.ClientId = configuration["AuthServer:ClientId"];
    options.ClientSecret = configuration["AuthServer:ClientSecret"];
    options.RequireHttpsMetadata = false;
    options.ResponseType = OpenIdConnectResponseType.CodeIdToken;
    options.SaveTokens = true;
    options.GetClaimsFromUserInfoEndpoint = true;
    options.Scope.Add("role");
    options.Scope.Add("email");
    options.Scope.Add("phone");
    options.Scope.Add("BackendAdminAppGateway");
    options.Scope.Add("IdentityService");
    options.Scope.Add("ProductService");
    options.ClaimActions.MapAbpClaimTypes();
});
```

```

- \* It adds "Cookies" authentication as the primary authentication type.
- \* "oidc" authentication is configured to use the AuthServer application as the authentication server.
- \* It requires the additional identity scopes **\*role\***, **\*email\*** and **\*phone\***.
- \* It requires the API resource scopes **\*BackendAdminAppGateway\***, **\*IdentityService\*** and **\*ProductService\*** because it will use these services as APIs.

IdentityServer client settings are stored inside the `appsettings.json` file:

```

```json
"AuthServer": {
    "Authority": "http://localhost:64999",
    "ClientId": "backend-admin-app-client",
    "ClientSecret": "1q2w3e*"
}
```

```

#### #### User Interface

The BackendAdminApp.Host project itself has not a single UI element/page. It is only used to serve UI pages of the Identity and Product Management modules.

``BackendAdminAppHostModule`` adds dependencies to ``AbpIdentityWebModule`` (`*/[Volo.Abp.Identity.Web](https://www.nuget.org/packages/Volo.Abp.Identity.Web)*` package) and ``ProductManagementWebModule`` (`*/ProductManagement.Web*` project) for that purpose.

A screenshot from the user management page:

```
![microservice-sample-backend-ui](./images/microservice-sample-backend-ui.png)
```

A screenshot from the permission management modal for a role:

![microservice-sample-backend-ui-permissions](./images/microservice-sample-backend-ui-permissions.png)

#### #### Using Microservices

Backend admin application uses the Identity and Product microservices for all operations, over the Backend Admin Gateway (`BackendAdminAppGateway.Host`).

##### ##### Remote End Point

``appsettings.json`` file contains the `RemoteServices` section to declare the remote service endpoint(s). Each microservice will normally have different endpoints. However, this solution uses the API Gateway pattern to provide a single endpoint for the applications:

```
```json
"RemoteServices": {
  "Default": {
    "BaseUrl": "http://localhost:65115/"
  }
}
````
```

``http://localhost:65115/`` is the URL of the `*BackendAdminAppGateway.Host*` project. It knows where are Identity and Product services are located.

##### ##### HTTP Clients

ABP application modules generally provides C# client libraries to consume services (APIs) easily (they generally uses the [Dynamic C# API Clients](./API/Dynamic-CSharp-API-Clients.md) feature of the ABP framework). That means if you need to consume Identity service API, you can reference to its client package and easily use the APIs by provided interfaces.

For that purpose, ``BackendAdminAppHostModule`` class declares dependencies for ``AbpIdentityHttpApiClientModule`` and ``ProductManagementHttpApiClientModule``.

Once you refer these client packages, you can directly inject an application service interface (e.g. ``IIdentityUserAppService``) and use its methods like a local method call. It actually invokes remote service calls over HTTP to the related service endpoint.

##### ##### Passing the Access Token

Since microservices requires authentication & authorization, each remote service call should contain an Authentication header. This header is obtained from the `access\_token` inside the current `HttpContext` for the current user. This is automatically done when you use the `Volo.Abp.Http.Client.IdentityModel` package. ``BackendAdminAppHostModule`` declares dependencies to this package and to the related ``AbpHttpClientIdentityModelModule`` class. It is integrated to the HTTP Clients explained above.

##### #### Dependencies

- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

#### ### Public Web Site (PublicWebSite.Host)

This is a public web site project that has a web blog and product list page.

#### #### Authentication

PublicWebSite can show blog posts and product list without login. If you login, you can also manage blogs. It redirects to the AuthServer for authentication. Once the user enters a correct username & password, the page is redirected to the public web site application again. Authentication configuration is setup in the `PublicWebHostModule` class:

```
```csharp
context.Services.AddAuthentication(options =>
{
    options.DefaultScheme = "Cookies";
    options.DefaultChallengeScheme = "oidc";
})
.AddCookie("Cookies", options =>
{
    options.Cookie.Expiration = TimeSpan.FromDays(365);
    options.ExpireTimeSpan = TimeSpan.FromDays(365);
})
.AddOpenIdConnect("oidc", options =>
{
    options.Authority = configuration["AuthServer:Authority"];
    options.ClientId = configuration["AuthServer:ClientId"];
    options.ClientSecret = configuration["AuthServer:ClientSecret"];
    options.RequireHttpsMetadata = false;
    options.ResponseType = OpenIdConnectResponseType.CodeIdToken;
    options.SaveTokens = true;
    options.GetClaimsFromUserInfoEndpoint = true;
    options.Scope.Add("role");
    options.Scope.Add("email");
    options.Scope.Add("phone");
    options.Scope.Add("PublicWebSiteGateway");
    options.Scope.Add("ProductService");
    options.Scope.Add("BloggingService");
    options.ClaimActions.MapAbpClaimTypes();
});
```

```

- It adds "Cookies" authentication as the primary authentication type.
- "oidc" authentication is configured to use the AuthServer application as the authentication server.
- It requires the additional identity scopes **\*role\***, **\*email\*** and **\*phone\***.
- It requires the API resource scopes **\*PublicWebSiteGateway\***, **\*BloggingService\*** and **\*ProductService\*** because it will use these services as APIs.

IdentityServer client settings are stored inside the `appsettings.json` file:

```
```json
"AuthServer": {
    "Authority": "http://localhost:64999",
    "ClientId": "public-website-client",
    "ClientSecret": "1q2w3e*"
}
````
```

#### #### User Interface

The PublicWebSite.Host project has a page to list products (`Pages/Products.cshtml`). It also uses the UI from the blogging module. `PublicWebSiteHostModule` adds dependencies to `BloggingWebModule`  
(\*[*Volo.Blogging.Web*](https://www.nuget.org/packages/Volo.Blogging.Web)\* package) for that purpose.

A screenshot from the Products page:

![microservice-sample-public-product-list](./images/microservice-sample-public-product-list.png)

#### #### Using Microservices

Public web site application uses the Blogging and Product microservices for all operations, over the Public Web Site Gateway (PublicWebSiteGateway.Host).

#### ##### Remote End Point

`appsettings.json` file contains the `RemoteServices` section to declare the remote service endpoint(s). Each microservice will normally have different endpoints. However, this solution uses the API Gateway pattern to provide a single endpoint for the applications:

```
```json
"RemoteServices": {
    "Default": {
        "BaseUrl": "http://localhost:64897/"
    }
}
````
```

`http://localhost:64897/` is the URL of the \*PublicWebSiteGateway.Host\* project. It knows where are Blogging and Product services are located.

#### ##### HTTP Clients

`PublicWebSiteHostModule` class declares dependencies for `BloggingHttpApiClientModule` and `ProductManagementHttpApiClientModule` to be able to use remote HTTP APIs for these services.

#### ##### Passing the Access Token

Just like explained in the Backend Admin Application section, Public Web Site project also uses the `AbpHttpClientIdentityModelModule` to pass `access\_token` to the calling services for authentication.

#### #### Dependencies

- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

#### ### Console Client Demo

Finally, the solution includes a very simple console application, named ConsoleClientDemo, that uses Identity and Product services by authenticating through the AuthServer. It uses the Internal Gateway (InternalGateway.Host) to perform HTTP API calls.

#### #### Remote Service Configuration

`RemoteService` configuration in the `appsettings.json` file is simple:

```
```json
"RemoteServices": {
  "Default": {
    "BaseUrl": "http://localhost:65129/"
  }
}
````
```

`http://localhost:65129/` is the URL of the Internal Gateway. All API calls to the services are performed over this URL.

#### #### Authentication (IdentityServer Client) Configuration

`appsettings.json` also has a configuration for the IdentityServer authentication:

```
```json
"IdentityClients": {
  "Default": {
    "GrantType": "client_credentials",
    "ClientId": "console-client-demo",
    "ClientSecret": "1q2w3e*",
    "Authority": "http://localhost:64999",
    "Scope": "InternalGateway IdentityService ProductService"
  }
}
````
```

This sample uses the `client\_credentials` grant type which requires a `ClientId` and `ClientSecret` for the authentication process. There are also [other grant types]([http://docs.identityserver.io/en/latest/topics/grant\\_types.html](http://docs.identityserver.io/en/latest/topics/grant_types.html)). For example, you

can use the following configuration to switch to the `password` (Resource Owner Password) grant type:

```
```json
"IdentityClients": {
  "Default": {
    "GrantType": "password",
    "ClientId": "console-client-demo",
    "ClientSecret": "1q2w3e*",
    "UserName": "admin",
    "UserPassword": "1q2w3E*",
    "Authority": "http://localhost:64999",
    "Scope": "InternalGateway IdentityService ProductService"
  }
}
````
```

Resource Owner Password requires a `UserName` & `UserPassword` in addition to client credentials. This grant type is useful to call remote services on behalf of a user.

`Scope` declares the APIs (and the gateway) to grant access. This application uses the Internal Gateway.

#### #### HTTP Client Dependencies

`ConsoleClientDemoModule` has dependencies to `AbpIdentityHttpApiClientModule` and `ProductManagementHttpApiClientModule` in order to use Identity and Product APIs. It also has `AbpHttpClientIdentityModelModule` dependency to authenticate via IdentityServer.

#### #### Using the Services

Using the services is straightforward. See the `ClientDemoService` class which simply injects `IIdentityUserAppService` and `IProductAppService` and uses them. This class also shows a manual HTTP call using an `HttpClient` object. See source code of the `ClientDemoService` for details.

#### ## API Gateways / BFFs (Backend for Frontend)

Gateways are used to provide a **single entry point** to the applications. In this way, an application only deal with a single service address (API endpoint) instead of a different addresses for each services. Gateways are also used for rate limiting, security, authentication, load balancing and many more requirements.

**Backend for Frontend** (BFF) is a common architectural pattern which offers to build a **dedicated and specialized** gateway for each different application / client type. This solution uses this pattern and has multiple gateways.

This solution uses the [Ocelot](<https://github.com/ThreeMammals/Ocelot>) library to build API Gateways. It's a widely accepted API Gateway library for ASP.NET Core.

#### ### Backend Admin Application Gateway (BackendAdminAppGateway.Host)

This is backend (server side API) for the "Backend Admin Application" (don't confuse about the naming; Backend Admin Application is a frontend web application actually, but used by system admins rather than regular users).

#### #### Authentication

This gateway uses IdentityServer `Bearer` authentication and configured like that:

```
```csharp
context.Services.AddAuthentication("Bearer")
    .AddIdentityServerAuthentication(options =>
{
    options.Authority = configuration["AuthServer:Authority"];
    options.ApiName = configuration["AuthServer:ApiName"];
    options.RequireHttpsMetadata = false;
    options.InboundJwtClaimTypeMap["sub"] = AbpClaimTypes.UserId;
    options.InboundJwtClaimTypeMap["role"] = AbpClaimTypes.Role;
    options.InboundJwtClaimTypeMap["email"] = AbpClaimTypes.Email;
    options.InboundJwtClaimTypeMap["email_verified"] = AbpClaimTypes.EmailVerified;
    options.InboundJwtClaimTypeMap["phone_number"] = AbpClaimTypes.PhoneNumber;
    options.InboundJwtClaimTypeMap["phone_number_verified"] =
        AbpClaimTypes.PhoneNumberVerified;
    options.InboundJwtClaimTypeMap["name"] = AbpClaimTypes.UserName;
});
````
```

`AddIdentityServerAuthentication` extension method comes from the [\[IdentityServer4.AccessTokenValidation\]\(https://www.nuget.org/packages/IdentityServer4.AccessTokenValidation\)](#) package, part of the IdentityServer4 project (see [\[its documentation\]\(http://docs.identityserver.io/en/latest/topics/apis.html\)](#)).

`ApiName` is the API which is being protected, `BackendAdminAppGateway` in this case. So, this solution defines gateways as API resources. Rest of the configuration is related to claims mapping (which is planned to be automated in next ABP versions). The configuration related to authentication in the `appsettings.json` is simple:

```
```json
"AuthServer": {
    "Authority": "http://localhost:64999",
    "ApiName": "BackendAdminAppGateway"
}
````
```

#### #### Ocelot Configuration

Ocelot needs to know the real URLs of the microservices to be able to redirect HTTP requests. The configuration for this gateway is like below:

```
```json
"ReRoutes": [
{
    "DownstreamPathTemplate": "/api/identity/{everything}",
    "UpstreamPathTemplate": "/{everything}"
}
]
````
```

```

 "DownstreamScheme": "http",
 "DownstreamHostAndPorts": [
 {
 "Host": "localhost",
 "Port": 63568
 }
],
 "UpstreamPathTemplate": "/api/identity/{everything}",
 "UpstreamHttpMethod": ["Put", "Delete", "Get", "Post"]
},
{
 "DownstreamPathTemplate": "/api/productManagement/{everything}",
 "DownstreamScheme": "http",
 "DownstreamHostAndPorts": [
 {
 "Host": "localhost",
 "Port": 60244
 }
],
 "UpstreamPathTemplate": "/api/productManagement/{everything}",
 "UpstreamHttpMethod": ["Put", "Delete", "Get", "Post"]
}
],
"GlobalConfiguration": {
 "BaseUrl": "http://localhost:65115"
}
```

```

`ReRoutes` is an array of URL mappings. `BaseUrl` in the `GlobalConfiguration` section is the URL of this gateway (Ocelot needs to know its own URL). See [[its own documentation](#)] (<https://ocelot.readthedocs.io/en/latest/features/configuration.html>) to better understand the configuration.

Ocelot is a finalizer ASP.NET Core middleware and should be written as the last item in the pipeline:

```

```csharp
app.UseOcelot().Wait();
```

```

It handles and redirects requests based on the configuration above.

ABP Configuration Endpoints

ABP provides some built-in APIs to get some configuration and information from the server. Examples:

- * `/api/abp/application-configuration` returns localization texts, permission and setting values (try <http://localhost:65115/api/abp/application-configuration> for this gateway).
- * `/Abp/ServiceProxyScript` returns dynamic javascript proxies to call services from a javascript client (try <http://localhost:65115/Abp/ServiceProxyScript> for this gateway).

These endpoints should be served by the gateway service, not by microservices. A microservice can only know permissions related to that microservice. But, once properly configured, gateway can aggregate permission values for multiple services as a single list which is more suitable for clients.

For this purpose, the ASP.NET Core pipeline was configured to handle some specific routes via MVC, instead of Ocelot. To make this possible, `MapWhen` extension method is used like that:

```
```csharp
app.MapWhen(ctx => ctx.Request.Path.ToString().StartsWith("/api/abp/") ||
 ctx.Request.Path.ToString().StartsWith("/Abp/"),
 app2 =>
{
 app2.UseConfiguredEndpoints();
});

app.UseOcelot().Wait();
````
```

This configuration uses standard MVC middleware when request path starts with `/api/abp/` or `/Abp/`.

Swagger

This gateway is configured to use the [swagger UI](<https://swagger.io/tools/swagger-ui/>), a popular tool to discover & test HTTP APIs. Normally, Ocelot does not support to show APIs on the swagger, because it can not know details of each microservice API. But it is possible when you follow ABP layered module architecture [best practices]([./Best-Practices/Index.md](#)).

`BackendAdminAppGatewayHostModule` adds dependency to `AbpIdentityHttpApiModule` (*[\[Volo.Abp.Identity.HttpApi\]](#)(<https://www.nuget.org/packages/Volo.Abp.Identity.HttpApi>)*) package and `ProductManagementHttpApiModule` (*[ProductManagement.HttpApi](#)* project) to include their HTTP API Controllers. In this way, swagger can discover them. While it references to the API layer, it does not reference to the implementation of application services, because they will be running in the related microservice endpoints and redirected by the Ocelot based on the request URL.

Anyway, when you open the URL `http://localhost:65115/swagger/index.html`, you will see APIs of all configured microservices.

Permission Management

Backend Admin Application provides a permission management UI (seen before) and uses this gateway to get/set permissions. Permission management API is hosted inside the gateway, instead of a separate service. This is a design decision, but it could be hosted as another microservice if you would like.

Dependencies

- **RabbitMQ** for messaging to other services.

- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

Public Web Site Gateway (PublicWebSiteGateway.Host)

This is backend (server side API gateway) for the "Public Web Site" application.

Authentication

This gateway uses IdentityServer `Bearer` authentication and configured like that:

```
```csharp
context.Services.AddAuthentication("Bearer")
 .AddIdentityServerAuthentication(options =>
{
 options.Authority = configuration["AuthServer:Authority"];
 options.ApiName = configuration["AuthServer:ApiName"];
 options.RequireHttpsMetadata = false;
 options.InboundJwtClaimTypeMap["sub"] = AbpClaimTypes.UserId;
 options.InboundJwtClaimTypeMap["role"] = AbpClaimTypes.Role;
 options.InboundJwtClaimTypeMap["email"] = AbpClaimTypes.Email;
 options.InboundJwtClaimTypeMap["email_verified"] = AbpClaimTypes.EmailVerified;
 options.InboundJwtClaimTypeMap["phone_number"] = AbpClaimTypes.PhoneNumber;
 options.InboundJwtClaimTypeMap["phone_number_verified"] =
 AbpClaimTypes.PhoneNumberVerified;
 options.InboundJwtClaimTypeMap["name"] = AbpClaimTypes.UserName;
});
```
`AddIdentityServerAuthentication` extension method comes from the [IdentityServer4.AccessTokenValidation](https://www.nuget.org/packages/IdentityServer4.AccessTokenValidation) package, part of the IdentityServer4 project (see [its documentation](http://docs.identityserver.io/en/latest/topics/apis.html)).
```

`ApiName` is the API which is being protected, `PublicWebSiteGateway` in this case. Rest of the configuration is related to claims mapping (which is planned to be automated in next ABP versions). The configuration related to authentication in the `appsettings.json` is simple:

```
```json
"AuthServer": {
 "Authority": "http://localhost:64999",
 "ApiName": "PublicWebSiteGateway"
}
```

```

Ocelot Configuration

Ocelot needs to know the real URLs of the microservices to be able to redirect HTTP requests. The configuration for this gateway is like below:

```
```json

```

```

"ReRoutes": [
 {
 "DownstreamPathTemplate": "/api/productManagement/{everything}",
 "DownstreamScheme": "http",
 "DownstreamHostAndPorts": [
 {
 "Host": "localhost",
 "Port": 60244
 }
],
 "UpstreamPathTemplate": "/api/productManagement/{everything}",
 "UpstreamHttpMethod": ["Put", "Delete", "Get", "Post"]
 },
 {
 "DownstreamPathTemplate": "/api/blogging/{everything}",
 "DownstreamScheme": "http",
 "DownstreamHostAndPorts": [
 {
 "Host": "localhost",
 "Port": 62157
 }
],
 "UpstreamPathTemplate": "/api/blogging/{everything}",
 "UpstreamHttpMethod": ["Put", "Delete", "Get", "Post"]
 }
],
"GlobalConfiguration": {
 "BaseUrl": "http://localhost:64897"
}
```

```

See [[its own documentation](#)] (<https://ocelot.readthedocs.io/en/latest/features/configuration.html>) to better understand the Ocelot configuration.

Other

See the "ABP Configuration Endpoints" and "Swagger" topics inside the "Backend Admin Application Gateway" section which are very similar for this gateway.

Dependencies

- **RabbitMQ** for messaging to other services.
- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

Internal Gateway (InternalGateway.Host)

This gateway is not a BFF. It is designed for inter-microservice communication and is not exposed publicly.

Authentication

This gateway uses IdentityServer `Bearer` authentication and configured like that:

```
```csharp
context.Services.AddAuthentication("Bearer")
 .AddIdentityServerAuthentication(options =>
{
 options.Authority = configuration["AuthServer:Authority"];
 options.ApiName = configuration["AuthServer:ApiName"];
 options.RequireHttpsMetadata = false;
 options.InboundJwtClaimTypeMap["sub"] = AbpClaimTypes.UserId;
 options.InboundJwtClaimTypeMap["role"] = AbpClaimTypes.Role;
 options.InboundJwtClaimTypeMap["email"] = AbpClaimTypes.Email;
 options.InboundJwtClaimTypeMap["email_verified"] = AbpClaimTypes.EmailVerified;
 options.InboundJwtClaimTypeMap["phone_number"] = AbpClaimTypes.PhoneNumber;
 options.InboundJwtClaimTypeMap["phone_number_verified"] =
 AbpClaimTypes.PhoneNumberVerified;
 options.InboundJwtClaimTypeMap["name"] = AbpClaimTypes.UserName;
});
```

```

`AddIdentityServerAuthentication` extension method comes from the [[IdentityServer4.AccessTokenValidation](#)] (<https://www.nuget.org/packages/IdentityServer4.AccessTokenValidation>) package, part of the IdentityServer4 project (see [[its documentation](#)] (<http://docs.identityserver.io/en/latest/topics/apis.html>)).

`ApiName` is the API which is being protected, `InternalGateway` in this case. Rest of the configuration is related to claims mapping (which is planned to be automated in next ABP versions). The configuration related to authentication in the `appsettings.json` is simple:

```
```json
"AuthServer": {
 "Authority": "http://localhost:64999",
 "ApiName": "InternalGateway"
}
```

```

Ocelot Configuration

Ocelot needs to know the real URLs of the microservices to be able to redirect HTTP requests. The configuration for this gateway is like below:

```
```json
"ReRoutes": [
 {
 "DownstreamPathTemplate": "/api/identity/{everything}",
 "DownstreamScheme": "http",
 "DownstreamHostAndPorts": [
 {
 "Host": "localhost",
 "Port": 63568
 }
]
 }
]
```

```

```

        }
    ],
    "UpstreamPathTemplate": "/api/identity/{everything}",
    "UpstreamHttpMethod": [ "Put", "Delete", "Get", "Post" ]
},
{
    "DownstreamPathTemplate": "/api/productManagement/{everything}",
    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
        {
            "Host": "localhost",
            "Port": 60244
        }
    ],
    "UpstreamPathTemplate": "/api/productManagement/{everything}",
    "UpstreamHttpMethod": [ "Put", "Delete", "Get", "Post" ]
},
{
    "DownstreamPathTemplate": "/api/blogging/{everything}",
    "DownstreamScheme": "http",
    "DownstreamHostAndPorts": [
        {
            "Host": "localhost",
            "Port": 62157
        }
    ],
    "UpstreamPathTemplate": "/api/blogging/{everything}",
    "UpstreamHttpMethod": [ "Put", "Delete", "Get", "Post" ]
}
],
"GlobalConfiguration": {
    "BaseUrl": "http://localhost:65129"
}
```

```

``ReRoutes`` configuration covers all microservices in the system. See [\[its own documentation\]](#) (<https://ocelot.readthedocs.io/en/latest/features/configuration.html>) to better understand the Ocelot configuration.

#### #### Other

See the "ABP Configuration Endpoints" and "Swagger" topics inside the "Backend Admin Application Gateway" section which are very similar for this gateway.

#### #### Dependencies

- **RabbitMQ** for messaging to other services.
- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

#### ## Microservices

Microservices are standalone HTTP APIs those implement the business of the system in a distributed manner.

- \* They are used by applications and other microservices through the gateways and HTTP APIs.
- \* They can raise or register to events in the system.
- \* They can communicate to each other via asynchronous messaging.

#### ### Identity Service (`IdentityService.Host`)

This service provides user and role management APIs.

#### #### Database

Shares the same database (`MsDemo_Identity`) with the `AuthServer` application.

#### #### Identity Module

This service actually just hosts the ABP Identity package/module. Does not include any API itself. In order to host it, adds the following dependencies:

- \* ``AbpIdentityHttpApiModule``  
(\*[\[Volo.Abp.Identity.HttpApi\]](#) (<https://www.nuget.org/packages/Volo.Abp.Identity.HttpApi>) package) to provide Identity APIs.
- \* ``AbpIdentityApplicationModule``  
(\*[\[Volo.Abp.Identity.Application\]](#) (<https://www.nuget.org/packages/Volo.Abp.Identity.Application>) package) to host the implementation of the application and domain layers of the module.
- \* ``AbpIdentityEntityFrameworkCoreModule``  
(\*[\[Volo.Abp.Identity.EntityFrameworkCore\]](#) (<https://www.nuget.org/packages/Volo.Abp.Identity.EntityFrameworkCore>) package) to use EF Core as database API.

See the [\[module architecture best practice guide\]](#) ([..../Best-Practices/Module-Architecture](#)) to understand the layering better.

#### #### Authentication

This microservice uses `IdentityServer` ``Bearer`` authentication and configured like that:

```
```csharp
context.Services.AddAuthentication("Bearer")
    .AddIdentityServerAuthentication(options =>
{
    options.Authority = configuration["AuthServer:Authority"];
    options.ApiName = configuration["AuthServer:ApiName"];
    options.RequireHttpsMetadata = false;
    options.InboundJwtClaimTypeMap["sub"] = AbpClaimTypes.UserId;
    options.InboundJwtClaimTypeMap["role"] = AbpClaimTypes.Role;
    options.InboundJwtClaimTypeMap["email"] = AbpClaimTypes.Email;
    options.InboundJwtClaimTypeMap["email_verified"] = AbpClaimTypes.EmailVerified;
    options.InboundJwtClaimTypeMap["phone_number"] = AbpClaimTypes.PhoneNumber;
    options.InboundJwtClaimTypeMap["phone_number_verified"] =
```

```
        AbpClaimTypes.PhoneNumberVerified;
        options.InboundJwtClaimTypeMap["name"] = AbpClaimTypes.UserName;
    });
```

```

```ApiName`` is the API which is being protected, `IdentityService` in this case. Rest of the configuration is related to claims mapping (which is planned to be automated in next ABP versions). The configuration related to authentication in the `appsettings.json` is simple:

```
```json
"AuthServer": {
 "Authority": "http://localhost:64999",
 "ApiName": "IdentityService"
}
```

```

Swagger

Swagger UI is configured and is the default page for this service. If you navigate to the URL `http://localhost:63568/`, you are redirected to the swagger page to see and test the API.

Dependencies

- **RabbitMQ** for messaging to other services.
- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

Blogging Service (BloggingService.Host)

This service hosts the blogging API.

Database

It has a dedicated MongoDB database (`MsDemo_Blogging`) to store blog and posts. It also uses the `MsDemo_Identity` SQL database for audit logs, permissions and settings. So, there are two connection strings in the `appsettings.json` file:

```
```json
"ConnectionStrings": {
 "Default": "Server=localhost;Database=MsDemo_Identity;Trusted_Connection=True",
 "Blogging": "mongodb://localhost/MsDemo_Blogging"
}
```

```

Blogging Module

This service actually just hosts the ABP Blogging package/module. Does not include any API itself. In order to host it, adds the following dependencies:

- `BloggingHttpApiModule`
(*[*Volo.Blogging.HttpApi*](https://www.nuget.org/packages/Volo.Blogging.HttpApi)* package) to provide Blogging APIs.
- `BloggingApplicationModule`
(*[*Volo.Blogging.Application*](https://www.nuget.org/packages/Volo.Blogging.Application)* package) to host the implementation of the application and domain layers of the module.
- `BloggingMongoDbModule`
(*[*Volo.Blogging.MongoDB*](https://www.nuget.org/packages/Volo.Abp.Identity.EntityFrameworkCore)* package) to use MongoDB as the database.

See the [module architecture best practice guide](../Best-Practices/Module-Architecture) to understand the layering better.

Authentication

This microservice uses IdentityServer `Bearer` authentication and configured like that:

```
```csharp
context.Services.AddAuthentication("Bearer")
 .AddIdentityServerAuthentication(options =>
{
 options.Authority = configuration["AuthServer:Authority"];
 options.ApiName = configuration["AuthServer:ApiName"];
 options.RequireHttpsMetadata = false;
 options.InboundJwtClaimTypeMap["sub"] = AbpClaimTypes.UserId;
 options.InboundJwtClaimTypeMap["role"] = AbpClaimTypes.Role;
 options.InboundJwtClaimTypeMap["email"] = AbpClaimTypes.Email;
 options.InboundJwtClaimTypeMap["email_verified"] = AbpClaimTypes.EmailVerified;
 options.InboundJwtClaimTypeMap["phone_number"] = AbpClaimTypes.PhoneNumber;
 options.InboundJwtClaimTypeMap["phone_number_verified"] =
 AbpClaimTypes.PhoneNumberVerified;
 options.InboundJwtClaimTypeMap["name"] = AbpClaimTypes.UserName;
});
```

```

`ApiName` is the API which is being protected, `BloggingService` in this case. Rest of the configuration is related to claims mapping (which is planned to be automated in next ABP versions). The configuration related to authentication in the `appsettings.json` is simple:

```
```json
"AuthServer": {
 "Authority": "http://localhost:64999",
 "ApiName": "BloggingService"
}
```

```

IdentityServer Client

This microservice also uses the Identity microservice API through the Internal Gateway, because it needs to query user details (username, email, phone, name and surname) in some

cases. So, it is also a client for the IdentityServer and defines a section in the `appsettings.json` file for that:

```
```json
"IdentityClients": {
 "Default": {
 "GrantType": "client_credentials",
 "ClientId": "blogging-service-client",
 "ClientSecret": "1q2w3e*",
 "Authority": "http://localhost:64999",
 "Scope": "InternalGateway IdentityService"
 }
}
````
```

Since it uses the Internal Gateway, it should also configure the remote endpoint of the gateway:

```
```json
"RemoteServices": {
 "Default": {
 "BaseUrl": "http://localhost:65129/",
 "UseCurrentAccessToken": "false"
 }
}
````
```

When you set `UseCurrentAccessToken` to `false`, ABP ignores the current `access_token` in the current `HttpContext` and authenticates to the AuthServer with the credentials defined above.

Why not using the token of the current user in the current request? Because, the user may not have required permissions on the Identity module, so it can not just pass the current authentication token directly to the Identity service. In addition, some of the blog service APIs are anonymous (not requires authenticated user), so in some cases there is no "current user" in the HTTP request. For these reasons, Blogging service should be defined as a client for the Identity service with its own credentials and permissions.

If you check the `AbpPermissionGrants` table in the `MsDemo_Identity` database, you can see the related permission for the `blogging-service-client`.

```
![microservice-sample-blogservice-permission-in-database](./images/microservice-sample-blogservice-permission-in-database.png)
```

Swagger

Swagger UI is configured and is the default page for this service. If you navigate to the URL `http://localhost:62157/`, you are redirected to the swagger page to see and test the API.

Dependencies

- **RabbitMQ** for messaging to other services.
- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

Product Service (ProductService.Host)

This service hosts the Product Management API.

Database & EF Core Migrations

It has a separated SQL database, named **MsDemo_ProductManagement**, for the product management module. It uses EF Core as the database provider and has a DbContext named `ProductServiceMigrationDbContext`:

```
```csharp
public class ProductServiceMigrationDbContext :
AbpDbContext<ProductServiceMigrationDbContext>
{
 public ProductServiceMigrationDbContext(
 DbContextOptions<ProductServiceMigrationDbContext> options
) : base(options)
 {

 }

 protected override void OnModelCreating(ModelBuilder modelBuilder)
 {
 base.OnModelCreating(modelBuilder);

 modelBuilder.ConfigureProductManagement();
 }
}
```
```

Actual model configuration is done inside the `modelBuilder.ConfigureProductManagement()` extension method. This project maintains the database schema using EF Core migrations.

Notice that this DbContext is only for database migrations. Product Management module has its own `DbContext` class that is used in the runtime (See `ProductManagementDbContext` class in the `ProductManagement.EntityFrameworkCore` project).

There are two connection strings in the `appsettings.json` file:

```
```json
"ConnectionStrings": {
 "Default": "Server=localhost;Database=MsDemo_Identity;Trusted_Connection=True",
 "ProductManagement":
"Server=localhost;Database=MsDemo_ProductManagement;Trusted_Connection=True"
}
```
```

``Default`` connection strings points to the `MsDemo_Identity` database that is used for audit logging, permission and setting stores. ``ProductManagement`` connection string is used by the product module.

Product Module

This service actually just hosts the Product Management module. Does not include any API itself. In order to host it, adds the following dependencies:

- ``ProductManagementHttpApiModule`` to provide product management APIs.
- ``ProductManagementApplicationModule`` to host the implementation of the application and domain layers of the module.
- ``ProductManagementEntityFrameworkCoreModule`` to use EF Core as database API.

See the [\[module architecture best practice guide\]](#)(`./Best-Practices/Module-Architecture`) to understand the layering better. See the Product Management module section below for more information about this module.

Authentication

This microservice uses IdentityServer ``Bearer`` authentication and configured like that:

```
```csharp
context.Services.AddAuthentication("Bearer")
 .AddIdentityServerAuthentication(options =>
{
 options.Authority = configuration["AuthServer:Authority"];
 options.ApiName = configuration["AuthServer:ApiName"];
 options.RequireHttpsMetadata = false;
 options.InboundJwtClaimTypeMap["sub"] = AbpClaimTypes.UserId;
 options.InboundJwtClaimTypeMap["role"] = AbpClaimTypes.Role;
 options.InboundJwtClaimTypeMap["email"] = AbpClaimTypes.Email;
 options.InboundJwtClaimTypeMap["email_verified"] = AbpClaimTypes.EmailVerified;
 options.InboundJwtClaimTypeMap["phone_number"] = AbpClaimTypes.PhoneNumber;
 options.InboundJwtClaimTypeMap["phone_number_verified"] =
 AbpClaimTypes.PhoneNumberVerified;
 options.InboundJwtClaimTypeMap["name"] = AbpClaimTypes.UserName;
});
```

```

``ApiName`` is the API which is being protected, ``ProductService`` in this case. Rest of the configuration is related to claims mapping (which is planned to be automated in next ABP versions). The configuration related to authentication in the ``appsettings.json`` is simple:

```
```json
"AuthServer": {
 "Authority": "http://localhost:64999",
 "ApiName": "ProductService"
}
```

```

Swagger

Swagger UI is configured and is the default page for this service. If you navigate to the URL `http://localhost:60244/`, you are redirected to the swagger page to see and test the API.

Dependencies

- **RabbitMQ** for messaging to other services.
- **Redis** for distributed/shared caching.
- **Elasticsearch** for storing logs.

Modules

ABP provides a strong infrastructure to make modular application development easier by providing services and architecture (see the [\[module development best practices guide\]](#)(./Best-Practices/Index.md)).

This solution demonstrate how to use [\[prebuilt application modules\]](#)(./Modules/Index.md) in a distributed architecture. The solution also includes a simple "Product Management" module to show the implementation of a well layered module example.

Product Management

Product Management is a module that consists of several layers and packages/projects:

[!\[microservice-sample-product-module-in-solution\]\(./images/microservice-sample-product-module-in-solution.png\)](#)

- * `ProductManagement.Domain.Shared` contains constants and types shared among all layers.
- * `ProductManagement.Domain` contains the domain logic and defines entities, domain services, domain events, business/domain exceptions.
- * `ProductManagement.Application.Contracts` contains application service interfaces and DTOs.
- * `ProductManagement.Application` contains the implementation of application services.
- * `ProductManagement.EntityFrameworkCore` contains DbConext and other EF Core related classes and configuration.
- * `ProductManagement.HttpApi` contains API Controllers.
- * `ProductManagement.HttpApi.Client` contains C# proxies to directly use the HTTP API remotely. Uses [\[Dynamic C# API Clients\]](#)(./API/Dynamic-CSharp-API-Clients.md) feature of the ABP framework.
- * `ProductManagement.Web` contains the UI elements (pages, scripts, styles... etc).

By the help of this layering, it is possible to use the same module as a package reference in a monolithic application or use as a service that runs in another server. It is possible to separate UI (Web) and API layers, so they run in different servers.

In this solution, Web layer runs in the Backend Admin Application while API layer is hosted by the Product microservice.

This tutorial will highlight some important aspects of the module. But, it's suggested to see the source code for a better understanding.

Domain Layer

`Product` is the main [Aggregate Root](..../Entities.md) of this module:

```
```csharp
public class Product : AuditedAggregateRoot<Guid>
{
 /// <summary>
 /// A unique value for this product.
 /// ProductManager ensures the uniqueness of it.
 /// It can not be changed after creation of the product.
 /// </summary>
 [NotNull]
 public string Code { get; private set; }

 [NotNull]
 public string Name { get; private set; }

 public float Price { get; private set; }

 public int StockCount { get; private set; }

 //...
}
```

All of its properties have private setters which prevents any direct change of the properties from out of the class. Product class ensures its own integrity and validity by its own constructors and methods.

It has two constructors:

```
```csharp
private Product()
{
    //Default constructor is needed for ORMs.
}

internal Product(
    Guid id,
    [NotNull] string code,
    [NotNull] string name,
    float price = 0.0f,
    int stockCount = 0)
{
    Check.NotNullOrWhiteSpace(code, nameof(code));

    if (code.Length >= ProductConsts.MaxCodeLength)
    {
        throw new ArgumentException(
            $"Product code can not be longer than {ProductConsts.MaxCodeLength}");
    }
}
```

```

        );
    }

    Id = id;
    Code = code;
    SetName(Check.NotNullOrWhiteSpace(name, nameof(name)));
    SetPrice(price);
    SetStockCountInternal(stockCount, triggerEvent: false);
}
````
```

Default (**\*\*parameterless\*\***) constructor is private and is not used in the application code. It is needed because most ORMs requires a parameterless constructor on deserializing entities while getting from the database.

Second constructor is **\*\*internal\*\*** that means it can only be used inside the domain layer. This enforces to use the `ProductManager` while creating a new `Product`. Because, `ProductManager` should implement a business rule on a new product creation. This constructor only requires the minimal required arguments to create a new product with some optional arguments. It checks some simple business rules to ensure that the entity is created as a valid product.

Rest of the class has methods to manipulate properties of the entity. Example:

```

```csharp
public Product SetPrice(float price)
{
    if (price < 0.0f)
    {
        throw new ArgumentException($"'{nameof(price)}' can not be less than 0.0!");
    }

    Price = price;
    return this;
}
````
```

`SetPrice` method is used to change the price of the product in a safe manner (by checking a validation rule).

`SetStockCount` is another method that is used to change stock count of a product:

```

```csharp
public Product SetStockCount(int stockCount)
{
    return SetStockCountInternal(stockCount);
}

private Product SetStockCountInternal(int stockCount, bool triggerEvent = true)
{
```

```

if (StockCount < 0)
{
    throw new ArgumentException($"nameof(stockCount) can not be less than 0!");
}

if (StockCount == stockCount)
{
    return this;
}

if (triggerEvent)
{
    AddDistributedEvent(
        new ProductStockCountChangedEto(
            Id, StockCount, stockCount
        )
    );
}

StockCount = stockCount;
return this;
}
```

```

This method also triggers a **distributed event** with the `ProductStockCountChangedEto` parameter (Eto is a conventional postfix stands for **E**vent **T**ransfer **O**bject, but not required) to notify listeners that stock count of a product has changed. Any subscriber can receive this event and perform an action based on that knowledge.

Events are distributed by RabbitMQ for this solution. But ABP is message broker independent by providing necessary abstractions (see the [Event Bus](../Event-Bus.md) document).

As said before, this module forces to always use the `ProductManager` to create a new `Product`. `ProductManager` is a simple domain service defined as shown:

```

```csharp
public class ProductManager : DomainService
{
    private readonly IRepository<Product, Guid> _productRepository;

    public ProductManager(IRepository<Product, Guid> productRepository)
    {
        _productRepository = productRepository;
    }

    public async Task<Product> CreateAsync(
        [NotNull] string code,
        [NotNull] string name,
        float price = 0.0f,
        int stockCount = 0)
    {
        if (StockCount < 0)
        {
            throw new ArgumentException($"nameof(stockCount) can not be less than 0!");
        }

        if (StockCount == stockCount)
        {
            return this;
        }

        if (triggerEvent)
        {
            AddDistributedEvent(
                new ProductStockCountChangedEto(
                    Id, StockCount, stockCount
                )
            );
        }

        StockCount = stockCount;
        return this;
    }
}
```

```

```

 {
 var existingProduct =
 await _productRepository.FirstOrDefaultAsync(p => p.Code == code);

 if (existingProduct != null)
 {
 throw new ProductCodeAlreadyExistsException(code);
 }

 return await _productRepository.InsertAsync(
 new Product(
 GuidGenerator.Create(),
 code,
 name,
 price,
 stockCount
)
);
 }
}
```

```

- * It checks if given code is used before. Throws `ProductCodeAlreadyExistsException` so.
- * If uses the `GuidGenerator` (`IGuidGenerator`) service to create a new `Guid`.
- * It inserts the entity to the repository.

So, with this design, uniqueness of the product code is guaranteed.

`ProductCodeAlreadyExistsException` is a domain/business exception defined as like below:

```

```csharp
public class ProductCodeAlreadyExistsException : BusinessException
{
 public ProductCodeAlreadyExistsException(string productCode)
 : base("PM:000001", $"A product with code {productCode} has already exists!")
 {

 }
}
```

```

`PM:000001` is a code for the exception type that is sent to the clients, so they can understand the error type. Not implemented for this case, but it is also possible to localize business exceptions. See the [exception handling documentation](../Exception-Handling.md).

Application Layer

Application layer of this module has two services:

- * `ProductAppService` is mainly used by the Backend Admin Application to manage (create, update, delete...) products. It requires permission to perform any operation.

* `PublicProductAppService` is used by the Public Web Site to show list of products to the visitors. It does not require any permission since most of the visitors are not logged in to the application.

Notice that; instead of putting two application service into the same project, it might be a better principle to have separated application layers per application. But we unified them for simplicity in this solution.

As an example, `ProductAppService` has the following method to update a product:

```
```csharp
[Authorize(ProductManagementPermissions.Products.Update)]
public async Task<ProductDto> UpdateAsync(Guid id, UpdateProductDto input)
{
 var product = await _productRepository.GetAsync(id);

 product.SetName(input.Name);
 product.SetPrice(input.Price);
 product.SetStockCount(input.StockCount);

 return ObjectMapper.Map<Product, ProductDto>(product);
}
```
```

* It defines the required permission (**ProductManagementPermissions.Products.Update** is a constant with value `ProductManagement.Update`) to perform this operation.
* Gets the id of the product and a DTO contains the values to update.
* Gets the related product entity from the repository.
* Uses the related methods (like `SetName`) of the `Product` class to change properties, because they are with private setters and the only way to change a value is to use an entity method.
* Returns an updated `ProductDto` to the client (client may need it for some reason) by using the [ObjectMapper] (./Object-To-Object-Mapping.md).

The implementation may vary based on the requirements. This implementation follows the [best practices offered here] (./Best-Practices/Application-Services.md).

Other Layers

See other layers from the source code.

Infrastructure

Messaging and RabbitMQ

Asynchronous Messaging is a key concept in distributed systems. It makes possible to communicate as a loosely coupled manner with fault tolerance. It does not require both sides to be online at the moment of messaging. So, it is a widely used communication pattern in microservice architecture.

Distributed Event Bus

Distributed Events (Event Bus) is a way of messaging where a service raise/trigger events while other services registers/listens to these events to be notified when an important event occurs. ABP makes distributed events easier to use by providing conventions, services and integrations.

You have seen that the `Product` class publishing an event using the following code line:

```
```csharp
AddDistributedEvent(new ProductStockCountChangedEto(Id, StockCount, stockCount));
````
```

`ProductStockCountChangedEto` was defined as shown below:

```
```csharp
[Serializable]
public class ProductStockCountChangedEto : EtoBase
{
 public Guid Id { get; }

 public int OldCount { get; set; }

 public int CurrentCount { get; set; }

 private ProductStockCountChangedEto()
 {
 //Default constructor is needed for deserialization.
 }

 public ProductStockCountChangedEto(Guid id, int oldCount, int currentCount)
 {
 Id = id;
 OldCount = oldCount;
 CurrentCount = currentCount;
 }
}
````
```

This object stores necessary information about the event. Another service can easily register to this event by implementing the `IDistributedEventHandler` interface with the generic `ProductStockCountChangedEto` parameter:

```
```csharp
public class MyHandler : IDistributedEventHandler<ProductStockCountChangedEto>
{
 public async Task HandleEventAsync(ProductStockCountChangedEto eventData)
 {
 var productId = eventData.Id;
 //...
 }
}
````
```

All the integration and communication are done by the ABP framework when you use the [\[Volo.Abp.EventBus.RabbitMQ\]](#) (<https://www.nuget.org/packages/Volo.Abp.EventBus.RabbitMQ>) package. If you need to publish events out of an entity, just inject the ``IDistributedEventBus`` and use the ``PublishAsync`` method.

See the [\[Event Bus\]](#) ([\(../Event-Bus.md\)](#)) documentation for more information about the distributed event system.

RabbitMQ Configuration

In this solution, [\[RabbitMQ\]](#) (<https://www.rabbitmq.com/>) is used for messaging & distributed events.

[\[Volo.Abp.EventBus.RabbitMQ\]](#) (<https://www.nuget.org/packages/Volo.Abp.EventBus.RabbitMQ>) package is required to integrate to the RabbitMQ for distributed event system. Then you need to add dependency to the ``AbpEventBusRabbitMqModule`` for your module. For example, ``ProductServiceHostModule`` declares this dependency.

``AbpEventBusRabbitMqModule`` gets configuration from the ``appsettings.json`` by default. For example, the Product Service has such a configuration:

```
```json
"RabbitMQ": {
 "Connections": {
 "Default": {
 "HostName": "localhost"
 }
 },
 "EventBus": {
 "ClientName": "MsDemo_ProductService",
 "ExchangeName": "MsDemo"
 }
}
````
```

Caching and Redis

A distributed system obviously needs to a distributed and shared cache, instead of isolated in-memory caches for each service.

[\[Redis\]](#) (<https://redis.io/>) is used as a distributed cache in this solution. The solution uses Microsoft's standard [\[Microsoft.Extensions.Caching.Redis\]](#) (<https://www.nuget.org/packages/Microsoft.Extensions.Caching.Redis>) package for integration. All applications and services uses Redis cache when you use and configure this package. See [\[Microsoft's documentation\]](#) (<https://docs.microsoft.com/en-us/aspnet/core/performance/caching/distributed>) for more.

The solution also uses the

[\[Microsoft.AspNetCore.DataProtection.StackExchangeRedis\]](#) (<https://www.nuget.org/packages/Microsoft.AspNetCore.DataProtection.StackExchangeRedis>) package to share data protection keys between applications and services over Redis cache.

Logging, Serilog, Elasticsearch and Kibana

This solution uses [Serilog] (<https://serilog.net/>) as a logging library. It is a widely used library which has many data source integrations including [Elasticsearch] (<https://www.elastic.co/products/elasticsearch>).

Logging configurations are done in `Program.cs` files using a code block similar to the given below:

```
```csharp
Log.Logger = new LoggerConfiguration()
 .MinimumLevel.Debug()
 .MinimumLevel.Override("Microsoft", LogEventLevel.Information)
 .Enrich.WithProperty("Application", "ProductService")
 .Enrich.FromLogContext()
 .WriteTo.File("Logs/logs.txt")
 .WriteTo.Elasticsearch(
 new ElasticsearchSinkOptions(new Uri(configuration["ElasticSearch:Url"]))
 {
 AutoRegisterTemplate = true,
 AutoRegisterTemplateVersion = AutoRegisterTemplateVersion.ESv6,
 IndexFormat = "msdemo-log-{0:yyyy.MM}"
 })
 .CreateLogger();
````
```

This configures multiple log target: File and Elasticsearch. `Application` property is set to `ProductService` for this example. This is a way of distinguishing the logs of multiple services in a single database. You can then query logs by the `Application` name.

Elasticsearch URL is read from the `appsettings.json` configuration file:

```
```json
"ElasticSearch": {
 "Url": "http://localhost:9200"
}
````
```

If you use Kibana, which is a Visualization tool that is well integrated to Elasticsearch, you can see some fancy UI about your logs:

![microservice-sample-kibana-2](./images/microservice-sample-kibana-2.png)

Figure - A dashboard that shows log and error counts by service/application.

![microservice-sample-kibana-1](./images/microservice-sample-kibana-1.png)

Figure - A list of log entries

Kibana URL is `http://localhost:5601/` by default.

Audit Logging

ABP provides automatic audit logging which saves every request in detail (who is the current user, what is the browser/client, what actions performed, which entities changed, even which properties of entities has been updated). See the [audit logging document](../Audit-Logging.md) for details.

All of the services and applications are configured to write audit logs. Audit logs are saved to the MsDemo_Identity SQL database. So, you can query all audit logs of all applications from a single point.

An Audit Log record has a `CorrelationId` property that can be used to track a request. When a service calls another service in a single web request, they both save audit logs with the same `CorrelationId`. See the `AbpAuditLogs` table in the database.

Multi-Tenancy

The solution has been configured to provide a [multi-tenant](../Multi-Tenancy.md) system, where each tenant can have their isolated users, roles, permissions and other data.

18 Books

18.1 Mastering ABP Framework

"<https://abp.io/books/mastering-abp-framework>"

18.2 Implementing Domain Driven Design

"<https://abp.io/books/implementing-domain-driven-design>"

19 Release Information

19.1 Upgrading

Upgrading the ABP Framework

This document explains how to upgrade your existing solution when a new ABP Framework version is published.

ABP UPDATE Command

ABP Framework & module ecosystem consist of hundreds of NuGet and NPM packages. It would be tedious to manually update all these packages to upgrade your application.

[ABP CLI](CLI.md) provides a handy command to update all the ABP related NuGet and NPM packages in your solution with a single command:

```
~~~~bash
```

```
abp update  
....
```

Run this command in the terminal while you are in the root folder of your solution.

› If your solution has the Angular UI, you probably have `aspnet-core` and `angular` folders in the solution. Run this command in the parent folder of these two folders.

Database Migrations

› Warning: Be careful if you are migrating your database since you may have data loss in some cases. Carefully check the generated migration code before executing it. It is suggested to take a backup of your current database.

When you upgrade to a new version, it is good to check if there is a database schema change and upgrade your database schema if your database provider is ****Entity Framework Core****:

- * Use `Add-Migration "Upgraded_To_Abp_4_1"` or a similar command in the Package Manager Console (PMC) to create a new migration (Set the `EntityFrameworkCore` as the Default project in the PMC and `DbMigrator` as the Startup Project in the Solution Explorer, in the Visual Studio).
- * Run the `DbMigrator` application to upgrade the database and seed the initial data.

If `Add-Migration` generates an empty migration, you can use `Remove-Migration` to delete it before executing the `DbMigrator`.

The Blog Posts & Guides

Whenever you upgrade your solution, it is strongly suggested to check the [ABP BLOG] (<https://blog.abp.io/>) to learn the new features and changes coming with the new version. We regularly publish posts and write these kind of changes.

Migration Guides

We prepare migration guides if the new version brings breaking changes for existing applications. See the [Migration Guides] (Migration-Guides/Index.md) page for all the guides.

Upgrading the Startup Template

Sometimes we introduce new features/changes that requires to ****make changes in the startup template****. We already implement the changes in the startup template for new applications. However, in some cases you need to manually make some minor changes in your existing solution.

It is not practical to document the necessary changes line by line. In this case, we suggest you to create an example solution, one with your existing version and one with the new version and compare them using a diff tool. You can [see this guide] (Migration-Guides/Upgrading-Startup-Template.md) to learn how you can do it using WinMerge application.

Semantic Versioning & Breaking Changes

We are working hard to keep the semantic versioning rules, so you don't get breaking changes for minor (feature) versions like 3.1, 3.2, 3.3...

However, there are some cases we may introduce breaking changes in feature versions too;

* ABP has many integration packages and sometimes the integrated libraries/frameworks releases major versions and makes breaking changes. In such cases, we carefully check these changes and decide to upgrade the integration package or not. If the impact of the change is relatively small, we update the integration package and explain the change in the release blog post. In such a case, if you've used this integration package, you should follow the instructions explained in the blog post. If the change may break many applications and not easy to fix, we decide to wait this upgrade until the next major ABP Framework release.

* Sometimes we have to make breaking change to fix a major bug or usage problem. In this case, we think that developer already can't properly use that feature, so no problem to fix it with a breaking change. In such cases, the feature will generally be a rarely used feature. Again, we try to keep the impact minimum.

Preview Releases & Nightly Builds

Preview releases and nightly builds can help you to try new features and adapt your solution earlier than a new stable release.

* [Preview releases](Previews.md) are typically published ~2 weeks before a minor (feature) version (our minor version development cycle is about ~4 weeks).
* [Nightly builds](Nightly-Builds.md) are published in every night (except weekends) from the development branch. That means you can try the previous day's development.

Refer to their documents to learn details about these kind of releases.

19.2 Official Packages

"<https://abp.io/packages>"

19.3 Preview Releases

Preview Releases

The preview versions are released ****~4 weeks before**** releasing a major or feature version of the ABP Framework. They are released for developers to try and provide feedback to have more stable versions.

Versioning of a preview release is like that:

- * 3.1.0-rc.1
- * 4.0.0-rc.1

More than one preview releases (like 3.1.0-rc.2 and 3.1.0-rc.3) might be published until the stable version (like 3.1.0).

Using the Preview Versions

Update the CLI

Before creating or updating an existing solution make sure to update the CLI to the latest preview version, for example:

```
```bash
dotnet tool update --global Volo.Abp.Cli --version 6.0.0-rc.2
```
```

New Solutions

To create a project for testing the preview version, you can select the "****preview****" option on the [download page](<https://abp.io/get-started>) or use the "**--preview**" parameter with the [ABP CLI](CLI.md) new command:

```
```bash
abp new Acme.BookStore --preview
```
```

This command will create a new project using the latest preview NuGet packages, NPM packages and the solution template. Whenever the stable version is released, you can switch to the stable version for your solution using the `abp switch-to-stable` command in the root folder of your solution.

Existing Solutions

If you already have a solution and want to use/test the latest preview version, use the following [ABP CLI](CLI.md) command in the root folder of your solution.

```
```bash
abp switch-to-preview
```
```

You can return back to the latest stable using the `abp switch-to-stable` command later.

```
```bash
abp switch-to-stable
```
```

Providing Feedback

You can open an issue on the [GitHub repository](<https://github.com/abpframework/abp/issues/new>), if you find a bug or want to provide any kind of feedback.

19.4 Nightly Builds

Nightly Builds

All framework & module packages are deployed to MyGet every night in weekdays. So, you can install the latest dev-brach builds to try out functionality prior to release.

Install & Uninstall Nightly Preview Packages

The latest version of nightly preview packages can be installed by the running below command in the root folder of the application:

```
```bash
abp switch-to-nightly
```
```

> Note that this command doesn't create a project with nightly preview packages. Instead, it switches package versions of a project with the nightly preview packages.

After this command, a new NuGet feed will be added to the `NuGet.Config` file of your project. Then, you can get the latest code of ABP Framework without waiting for the next release.

> You can check the [abp-nightly gallery](<https://www.myget.org/gallery/abp-nightly>) to see the all nightly preview packages.

If you're using the ABP Framework nightly preview packages, you can switch back to stable version using this command:

```
```bash
abp switch-to-stable
```
```

ABP nightly NuGet feed is [<https://www.myget.org/F/abp-nightly/api/v3/index.json>] (<https://www.myget.org/F/abp-nightly/api/v3/index.json>).

See the [ABP CLI documentation](./CLI.md) for more information.

19.5 Road Map

ABP Framework Road Map

This document provides a road map, release schedule and planned features for the ABP Framework.

Next Versions

v8.0

The next version will be 8.0 and planned to release the stable 8.0 version in December, 2023. We will be mostly working on the following topics:

- * Enabling nullable annotations for all projects
([#16610] (<https://github.com/abpframework/abp/issues/16610>))
- * Upgrade to .NET 8.0 ([#17355] (<https://github.com/abpframework/abp/issues/17355>))
- * Use NoTracking for readonly repositories for EF Core
([#597] (<https://github.com/abpframework/abp/issues/597>))
- * Support mapping collection of objects for custom object mappers
([#94] (<https://github.com/abpframework/abp/issues/94>))
- * Improvements on the existing features and provide more guides.

See the [8.0 milestone] (<https://github.com/abpframework/abp/milestone/88>) for all the issues we've planned to work on.

Backlog Items

The **Next Versions** section above shows the main focus of the planned versions. However, in each release we add new features to the core framework and the [application modules] (Modules/Index.md).

Here is a list of major items in the backlog we are considering working on in the next versions.

- * [#6655] (<https://github.com/abpframework/abp/pull/6655>) / Use Typescript for the MVC UI
- * [#236] (<https://github.com/abpframework/abp/issues/236>) / Resource based authorization system
- * [#2882] (<https://github.com/abpframework/abp/issues/2882>) / Providing a gRPC integration infrastructure (while it is [already possible] (<https://github.com/abpframework/abp-samples/tree/master/GrpcDemo>) to create or consume gRPC endpoints for your application, we plan to create endpoints for the [standard application modules] (<https://docs.abp.io/en/abp/latest/Modules/Index>))
- * [#57] (<https://github.com/abpframework/abp/issues/57>) / Built-in CQRS infrastructure
- * [#2532] (<https://github.com/abpframework/abp/issues/2532>) / CosmosDB integration with EF Core and MongoDB API
- * [#4223] (<https://github.com/abpframework/abp/issues/4223>) / WebHook system
- * [#162] (<https://github.com/abpframework/abp/issues/162>) / Azure ElasticDB Integration for multitenancy
- * [#2296] (<https://github.com/abpframework/abp/issues/2296>) / Feature toggling infrastructure
- * [#16342] (<https://github.com/abpframework/abp/issues/16342>) / CmsKit: Meta information for SEO
- * [#16260] (<https://github.com/abpframework/abp/issues/16260>) / GCP Blob Storage Provider
- * [#15932] (<https://github.com/abpframework/abp/issues/15932>) / Introduce ABP Diagnostics Module
- * [#16756] (<https://github.com/abpframework/abp/issues/16756>) / Blob Storing – Provider configuration UI
- * [#16744] (<https://github.com/abpframework/abp/issues/16744>) / State Management API

You can always check the milestone planning and the prioritized backlog issues on [the GitHub repository] (<https://github.com/abpframework/abp/milestones>) for a detailed road

map. The backlog items are subject to change. We are adding new items and changing priorities based on the community feedbacks and goals of the project.

Feature Requests

Vote for your favorite feature on the related GitHub issues (and write your thoughts). You can create an issue on [[the GitHub repository](#)] (<https://github.com/abpframework/abp>) for your feature requests, but first search in the existing issues.

19.6 Migration Guides

ABP Framework Migration Guides

The following documents explain how to migrate your existing ABP applications. We write migration documents only if you need to take an action while upgrading your solution. Otherwise, you can easily upgrade your solution using the [[abp update command](#)] (./Upgrading.md).

- [[7.3 to 7.4](#)] (Abp-7_4.md)
- [[7.2 to 7.3](#)] (Abp-7_3.md)
- [[7.1 to 7.2](#)] (Abp-7_2.md)
- [[7.0 to 7.1](#)] (Abp-7_1.md)
- [[6.0 to 7.0](#)] (Abp-7_0.md)
- [[5.3 to 6.0](#)] (Abp-6_0.md)
- [[5.2 to 5.3](#)] (Abp-5_3.md)
- [[5.1 to 5.2](#)] (Abp-5_2.md)
- [[4.x to 5.0](#)] (Abp-5_0.md)
- [[4.2 to 4.3](#)] (Abp-4_3.md)
- [[4.x to 4.2](#)] (Abp-4_2.md)
- [[3.3.x to 4.0](#)] (Abp-4_0.md)
- [[2.9.x to 3.0](#)] (./UI/Angular/Migration-Guide-v3.md)

20 API Documentation

"{[ApiDocumentationUrl](#)}"

21 Contribution Guide

Contribution Guide

ABP is an [[open source](#)] (<https://github.com/abpframework>) and community driven project. This guide is aims to help anyone wants to contribute to the project.

ABP Community Website

If you want to write ****articles**** or ****how to guides**** related to the ABP Framework and ASP.NET Core, please submit your article to the [\[community.abp.io\]](https://community.abp.io) (<https://community.abp.io/>) website.

Code Contribution

You can always send pull requests to the GitHub repository.

- [Fork] (<https://docs.github.com/en/free-pro-team@latest/github/getting-started-with-github/fork-a-repo>) the [\[ABP repository\]](https://github.com/abpframework/abp) (<https://github.com/abpframework/abp/>) from GitHub.

- Build the repository using the `'/build/build-all.ps1 -f'` for one time.
- Make the necessary changes, including unit/integration tests.
- Send a pull request.

> When you open a solution in Visual Studio, you may need to execute `'dotnet restore'` in the root folder of the solution for one time, after it is fully opened in the Visual Studio. This is needed since VS can't properly resolves local references to projects out of the solution.

GitHub Issues

Before making any change, please discuss it on the [\[Github issues\]](https://github.com/abpframework/abp/issues) (<https://github.com/abpframework/abp/issues>). In this way, no other developer will work on the same issue and your PR will have a better chance to be accepted.

Bug Fixes & Enhancements

You may want to fix a known bug or work on a planned enhancement. See [\[the issue list\]](https://github.com/abpframework/abp/issues) (<https://github.com/abpframework/abp/issues>) on Github.

Feature Requests

If you have a feature idea for the framework or modules, [\[create an issue\]](https://github.com/abpframework/abp/issues/new) (<https://github.com/abpframework/abp/issues/new>) on Github or attend to an existing discussion. Then you can implement it if it's embraced by the community.

Document Translation

You may want to translate the complete [\[documentation\]](https://docs.abp.io) (<https://docs.abp.io>) (including this one) to your mother language. If so, follow these steps:

- * Clone the [\[ABP repository\]](https://github.com/abpframework/abp) ([https://github.com/abpframework/abp/](https://github.com/abpframework/abp)) from Github.
- * To add a new language, create a new folder inside the [\[docs\]](https://github.com/abpframework/abp/tree/master/docs) (<https://github.com/abpframework/abp/tree/master/docs>) folder. Folder names can be "en", "es", "fr", "tr" and so on based on the language (see [\[all culture codes\]](https://msdn.microsoft.com/en-us/library/hh441729.aspx) (<https://msdn.microsoft.com/en-us/library/hh441729.aspx>)).
- * Get the [\["en" folder\]](https://github.com/abpframework/abp/tree/master/docs/en) (<https://github.com/abpframework/abp/tree/master/docs/en>) as a reference for the file names and folder structure. Keep the same naming if you are translating the same documentation.
- * Send a pull request (PR) once you translate any document. Please translate documents & send PRs one by one. Don't wait to finish translations for all documents.

There are some fundamental documents need to be translated before publishing a language on the [ABP documentation web site](<https://docs.abp.io>) :

- * Index (Home)
- * Getting Started
- * Web Application Development Tutorial

A new language is published after these minimum translations have been completed.

Resource Localization

ABP framework has a flexible [localization system](./Localization.md). You can create localized user interfaces for your own application.

In addition to that, the framework and the [pre-build modules](<https://docs.abp.io/en/abp/latest/Modules/Index>) have localized texts. As an example, see [the localization texts for the Volo.Abp.UI package](<https://github.com/abpframework/abp/blob/master/framework/src/Volo.Abp.UI/Localization/Resources/AbpUi/en.json>).

Using the "abp translate" command

This is the recommended approach, since it automatically finds all missing texts for a specific culture and lets you to translate in one place.

- * Clone the [ABP repository](<https://github.com/abpframework/abp/>) from Github.
- * Install the [ABP CLI](<https://docs.abp.io/en/abp/latest/CLI>) if you haven't installed before.
- * Run `abp translate -c <culture-name>` command for your language in the root folder of the abp repository. For example, use `abp translate -c fr` for French. Check [this document](<https://docs.microsoft.com/en-us/bingmaps/rest-services/common-parameters-and-types/supported-culture-codes>) to find the culture code for your language.
- * This command creates a file in the same folder, named `abp-translation.json`. Open this file in your favorite editor and fill the missing text values.
- * Once you done the translation, use `abp translate -a` command to apply changes to the related files.
- * Send a pull request on GitHub.

Manual Translation

If you want to make a change on a specific resource file, you can find the file yourself, make the necessary change (or create a new file for your language) and send a pull request on GitHub.

Bug Report

If you find any bug, please [create an issue on the Github repository](<https://github.com/abpframework/abp/issues/new>).

Setup Frontend Development Environment

[How to contribute to abp.io as a frontend developer] (How-to-Contribute-abp.io-as-a-frontend-developer.md)

See Also

- * [ABP Community Talks 2022.4: How can you contribute to the open source ABP Framework?] (<https://www.youtube.com/watch?v=Wz4Z-0-YoPg&list=PLsNcLT2aHJcOsPustEkzG6Dywi08eh01B>)