The software architecture

based on [Domain Driven Design] principles and patterns

Contents

[1 Quick Start 4](#_Toc143006329)

[1.1 With single-layer solution 4](#_Toc143006330)

[1.2 With layered architecture 4](#_Toc143006331)

[2 Getting Started 4](#_Toc143006332)

[2.1 Web Application - Layered Architecture 4](#_Toc143006333)

[2.1.1 1: Setup Your Development Environment 4](#_Toc143006334)

[2.1.2 2: Creating a New Solution 4](#_Toc143006335)

[2.1.3 3: Running the Solution 4](#_Toc143006336)

[2.2 Web Application - Single-Layered Architecture 4](#_Toc143006337)

[2.2.1 1: Setup Your Development Environment 4](#_Toc143006338)

[2.2.2 2: Creating a New Solution 4](#_Toc143006339)

[2.2.3 3: Running the Solution 4](#_Toc143006340)

[2.3 Console Application 4](#_Toc143006341)

[2.4 WPF Application 4](#_Toc143006342)

[2.5 MAUI 4](#_Toc143006343)

[2.6 Empty Web Project 4](#_Toc143006344)

[3 Tutorials 4](#_Toc143006345)

[3.1 Web Application Development 4](#_Toc143006346)

[3.1.1 1: Creating the Server Side 4](#_Toc143006347)

[3.1.2 2: The Book List Page 4](#_Toc143006348)

[3.1.3 3: Creating, Updating and Deleting Books 4](#_Toc143006349)

[3.1.4 4: Integration Tests 4](#_Toc143006350)

[3.1.5 5: Authorization 4](#_Toc143006351)

[3.1.6 6: Authors: Domain layer 4](#_Toc143006352)

[3.1.7 7: Authors: Database Integration 4](#_Toc143006353)

[3.1.8 8: Authors: Application Layer 4](#_Toc143006354)

[3.1.9 9: Authors: User Interface 4](#_Toc143006355)

[3.1.10 10: Book to Author Relation 4](#_Toc143006356)

[3.2 Community Articles 4](#_Toc143006357)

[3.3 Migrating from the ASP.NET Boilerplate 4](#_Toc143006358)

[4 CLI 4](#_Toc143006359)

[4.1 Examples for the new command 4](#_Toc143006360)

[5 Startup Templates 5](#_Toc143006361)

[5.1 Overall 5](#_Toc143006362)

[5.2 Application 5](#_Toc143006363)

[5.3 Application (Single Layer 5](#_Toc143006364)

[5.4 Module 5](#_Toc143006365)

[5.5 Console 5](#_Toc143006366)

[5.6 WPF 5](#_Toc143006367)

[6 Fundamentals 5](#_Toc143006368)

[6.1 Application Startup 5](#_Toc143006369)

[6.2 Authorization 5](#_Toc143006370)

[6.3 Caching 5](#_Toc143006371)

[6.3.1 Entity Cache 5](#_Toc143006372)

[6.3.2 Redis Cache 5](#_Toc143006373)

[6.4 Configuration 5](#_Toc143006374)

[6.4.1 Connection Strings 5](#_Toc143006375)

[6.5 Dependency Injection 5](#_Toc143006376)

[6.5.1 AutoFac Integration 5](#_Toc143006377)

[6.6 Exception Handling 5](#_Toc143006378)

[6.7 Localization 5](#_Toc143006379)

[6.8 Logging 5](#_Toc143006380)

[6.9 Object Extensions 5](#_Toc143006381)

[6.10 Options 5](#_Toc143006382)

[6.11 Settings 5](#_Toc143006383)

[6.12 Validation 5](#_Toc143006384)

[6.12.1 FluentValidation Integration 5](#_Toc143006385)

[7 Infrastructure 5](#_Toc143006386)

[7.1 Audit Logging 5](#_Toc143006387)

[7.2 Background Jobs 5](#_Toc143006388)

[7.2.1 Hangfire Integration 5](#_Toc143006389)

[7.2.2 RabbitMQ Integration 5](#_Toc143006390)

[7.2.3 Quartz Integration 5](#_Toc143006391)

[7.3 Background Workers 5](#_Toc143006392)

[7.3.1 Quartz Integration 5](#_Toc143006393)

[7.3.2 Hangfire Integration 6](#_Toc143006394)

[7.4 BLOB Storing 6](#_Toc143006395)

[7.4.1 BLOB Storing System 6](#_Toc143006396)

[7.4.2 Storage Providers 6](#_Toc143006397)

[7.5 Cancellation Token Provider 6](#_Toc143006398)

[7.6 CSRF/XSRF & Anti Forgery 6](#_Toc143006399)

[7.7 Concurrency Check 6](#_Toc143006400)

[7.8 Current User 6](#_Toc143006401)

[7.9 Data Filtering 6](#_Toc143006402)

[7.10 Data Seeding 6](#_Toc143006403)

[7.11 Distributed Locking 6](#_Toc143006404)

[7.12 Email Sending 6](#_Toc143006405)

[7.12.1 Email Sending System 6](#_Toc143006406)

[7.12.2 MailKit Integration 6](#_Toc143006407)

[7.13 Event Bus 6](#_Toc143006408)

[7.13.1 Overall 6](#_Toc143006409)

[7.13.2 Local Event Bus 6](#_Toc143006410)

[7.13.3 Distributed Event Bus 6](#_Toc143006411)

[7.14 Features 6](#_Toc143006412)

[7.15 Global Features 6](#_Toc143006413)

[7.16 GUID Generation 6](#_Toc143006414)

[7.17 Image Manipulation 6](#_Toc143006415)

[7.18 JSON 6](#_Toc143006416)

[7.19 Object to Object Mapping 6](#_Toc143006417)

[7.20 Simple State Checker 6](#_Toc143006418)

[7.21 SMS Sending 7](#_Toc143006419)

[7.22 String Encryption 7](#_Toc143006420)

[7.22.1 Text Templating 7](#_Toc143006421)

[7.23 Timing 7](#_Toc143006422)

[7.24 Virtual File System 7](#_Toc143006423)

[8 Architecture 7](#_Toc143006424)

[8.1 Modularity 7](#_Toc143006425)

[8.1.1 Basics 7](#_Toc143006426)

[8.1.2 Plug-In Modules 7](#_Toc143006427)

[8.1.3 Best Practices 7](#_Toc143006428)

[8.1.4 Customizing/Extending Modules 7](#_Toc143006429)

[8.2 Domain Driven Design 7](#_Toc143006430)

[8.2.1 Overall 7](#_Toc143006431)

[8.2.2 Domain Layer 7](#_Toc143006432)

[8.2.3 Application Layer 8](#_Toc143006433)

[8.2.4 E-Book: Implementing DDD 8](#_Toc143006434)

[8.3 Multi Tenancy 8](#_Toc143006435)

[8.4 Microservices 8](#_Toc143006436)

[9 API 8](#_Toc143006437)

[9.1 ABP Endpoints 8](#_Toc143006438)

[9.1.1 Application Configuration 8](#_Toc143006439)

[9.1.2 Application Localization 8](#_Toc143006440)

[9.2 API Versioning 8](#_Toc143006441)

[9.3 Auto API Controllers 8](#_Toc143006442)

[9.4 Dynamic C# API Clients 8](#_Toc143006443)

[9.5 Integration Services 8](#_Toc143006444)

[9.6 Static C# API Clients 8](#_Toc143006445)

[9.7 Swagger Integration 8](#_Toc143006446)

[10 User Interface 8](#_Toc143006447)

[10.1 MVC / Razor Pages 8](#_Toc143006448)

[10.1.1 Overall 8](#_Toc143006449)

[10.1.2 Navigation / Menus 8](#_Toc143006450)

[10.1.3 Forms & Validation 8](#_Toc143006451)

[10.1.4 Modals 8](#_Toc143006452)

[10.1.5 Data Tables 8](#_Toc143006453)

[10.1.6 Auto-Complete Select 8](#_Toc143006454)

[10.1.7 Page Alerts 8](#_Toc143006455)

[10.1.8 Dynamic JavaScript API Client Proxies 8](#_Toc143006456)

[10.1.9 Static JavaScript API Client Proxies 8](#_Toc143006457)

[10.1.10 Client Side Package Management 8](#_Toc143006458)

[10.1.11 Bundling & Minification 8](#_Toc143006459)

[10.1.12 Tag Helpers 8](#_Toc143006460)

[10.1.13 Widgets 9](#_Toc143006461)

[10.1.14 Toolbars 9](#_Toc143006462)

[10.1.15 Page Header 9](#_Toc143006463)

[10.1.16 Branding 9](#_Toc143006464)

[10.1.17 Layout Hooks 9](#_Toc143006465)

[10.1.18 Testing 9](#_Toc143006466)

[10.1.19 Theming 9](#_Toc143006467)

[10.1.20 JavaScript API 9](#_Toc143006468)

[10.1.21 Customize/Extend the UI 9](#_Toc143006469)

[10.1.22 Security 9](#_Toc143006470)

[10.2 Blazor 9](#_Toc143006471)

[10.2.1 Overall 9](#_Toc143006472)

[10.2.2 Navigation / Menu 9](#_Toc143006473)

[10.2.3 Localization 9](#_Toc143006474)

[10.2.4 Theming 9](#_Toc143006475)

[10.2.5 Security 10](#_Toc143006476)

[10.2.6 Services 10](#_Toc143006477)

[10.2.7 Other Components 10](#_Toc143006478)

[10.2.8 Settings 10](#_Toc143006479)

[10.2.9 Error Handling 10](#_Toc143006480)

[10.2.10 Customization / Overriding Components 10](#_Toc143006481)

[10.2.11 Global Scripts & Styles 10](#_Toc143006482)

[10.2.12 Global Features 10](#_Toc143006483)

[10.2.13 Routing 10](#_Toc143006484)

[10.2.14 PWA Configuration 10](#_Toc143006485)

[10.2.15 Layout Hooks 10](#_Toc143006486)

[10.3 Angular 10](#_Toc143006487)

[10.3.1 Quick Start 10](#_Toc143006488)

[10.3.2 Development 10](#_Toc143006489)

[10.3.3 Core Functionality 10](#_Toc143006490)

[10.3.4 Utilities 11](#_Toc143006491)

[10.3.5 Customization 11](#_Toc143006492)

[10.3.6 Components 12](#_Toc143006493)

[10.4 React Native 12](#_Toc143006494)

[10.4.1 Getting Started 12](#_Toc143006495)

[10.5 Common 12](#_Toc143006496)

[11 Data Access 12](#_Toc143006497)

[11.1 Overall 12](#_Toc143006498)

[11.2 Entity Framework Core 12](#_Toc143006499)

[11.2.1 Database Migrations 12](#_Toc143006500)

[11.2.2 Switch DBMS 12](#_Toc143006501)

[11.3 MongoDB 12](#_Toc143006502)

[11.4 Dapper 12](#_Toc143006503)

[12 Real Time 12](#_Toc143006504)

[12.1 SignalR Integration 12](#_Toc143006505)

[13 Dapr Integration 12](#_Toc143006506)

[14 Testing 12](#_Toc143006507)

[15 Deployment 13](#_Toc143006508)

[15.1 Configuring OpenIddict 13](#_Toc143006509)

[15.2 Configuring for Production 13](#_Toc143006510)

[15.3 Deploying to a Clustered Environment 13](#_Toc143006511)

[15.4 Distributed / Microservice Solutions 13](#_Toc143006512)

[15.5 Optimizing for Production 13](#_Toc143006513)

[16 Application Modules 13](#_Toc143006514)

[16.1 Overall 13](#_Toc143006515)

[16.2 Account 13](#_Toc143006516)

[16.3 Audit Logging 13](#_Toc143006517)

[16.4 Background Jobs 13](#_Toc143006518)

[16.5 CMS Kit 13](#_Toc143006519)

[16.6 Docs 13](#_Toc143006520)

[16.7 Feature Management 13](#_Toc143006521)

[16.8 Identity 13](#_Toc143006522)

[16.9 IdentityServer 13](#_Toc143006523)

[16.9.1 IdentityServer Migration Guide 13](#_Toc143006524)

[16.10 OpenIddict 13](#_Toc143006525)

[16.10.1 OpenIddict Migration Guide 13](#_Toc143006526)

[16.11 Permission Management 13](#_Toc143006527)

[16.12 Setting Management 13](#_Toc143006528)

[16.13 Tenant Management 13](#_Toc143006529)

[16.14 Virtual File Explorer 13](#_Toc143006530)

[16.15 Common 13](#_Toc143006531)

[16.15.1 Database Tables 13](#_Toc143006532)

[17 Samples 13](#_Toc143006533)

[17.1 All Samples 13](#_Toc143006534)

[17.2 eShopOnAbp 13](#_Toc143006535)

[17.3 EventHub 13](#_Toc143006536)

[17.4 Microservice Demo (legacy) 13](#_Toc143006537)

[18 Books 13](#_Toc143006538)

[18.1 Mastering ABP Framework 14](#_Toc143006539)

[18.2 Implementing Domain Driven Design 14](#_Toc143006540)

[19 Release Information 14](#_Toc143006541)

[19.1 Upgrading 14](#_Toc143006542)

[19.2 Official Packages 14](#_Toc143006543)

[19.3 Preview Releases 14](#_Toc143006544)

[19.4 Nightly Builds 14](#_Toc143006545)

[19.5 Road Map 14](#_Toc143006546)

[19.6 Migration Guides 14](#_Toc143006547)

[20 Contribution Guide 14](#_Toc143006548)

# 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.

## 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" />

}

<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

</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 }}}%}

</li>

</ul>

</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.

{{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).

## With layered architecture

# Getting Started

## Web Application - Layered Architecture

### 1: Setup Your Development Environment

### 2: Creating a New Solution

### 3: Running the Solution

## Web Application - Single-Layered Architecture

### 1: Setup Your Development Environment

### 2: Creating a New Solution

### 3: Running the Solution

## Console Application

## WPF Application

## MAUI

## Empty Web Project

# Tutorials

## Web Application Development

### 1: Creating the Server Side

### 2: The Book List Page

### 3: Creating, Updating and Deleting Books

### 4: Integration Tests

### 5: Authorization

### 6: Authors: Domain layer

### 7: Authors: Database Integration

### 8: Authors: Application Layer

### 9: Authors: User Interface

### 10: Book to Author Relation

## Community Articles

## Migrating from the ASP.NET Boilerplate

# CLI

## Examples for the new command

# Startup Templates

## Overall

## Application

## Application (Single Layer

## Module

## Console

## WPF

# Fundamentals

## Application Startup

## Authorization

## Caching

### Entity Cache

### Redis Cache

## Configuration

### Connection Strings

## Dependency Injection

### AutoFac Integration

## Exception Handling

## Localization

## Logging

## Object Extensions

## Options

## Settings

## Validation

### FluentValidation Integration

# Infrastructure

## Audit Logging

## Background Jobs

### Hangfire Integration

### RabbitMQ Integration

### Quartz Integration

## Background Workers

### Quartz Integration

### Hangfire Integration

## BLOB Storing

### BLOB Storing System

### Storage Providers

#### File System Provider

#### Database Provider

#### Azure Provider

#### Aliyun Provider

#### Minio Provider

#### AWS Provider

#### Create a Custom Provider

## Cancellation Token Provider

## CSRF/XSRF & Anti Forgery

## Concurrency Check

## Current User

## Data Filtering

## Data Seeding

## Distributed Locking

## Email Sending

### Email Sending System

### MailKit Integration

## Event Bus

### Overall

### Local Event Bus

### Distributed Event Bus

#### Azure Service Bus Integration

#### RabbitMQ Integration

#### Kafka Integration

#### Rebus Integration

## Features

## Global Features

## GUID Generation

## Image Manipulation

## JSON

## Object to Object Mapping

## Simple State Checker

## SMS Sending

## String Encryption

### Text Templating

#### Razor Integration

#### Scriban Integration

## Timing

## Virtual File System

# Architecture

## Modularity

### Basics

### Plug-In Modules

### Best Practices

#### Overall

#### Module Architecture

#### Domain Layer

##### Entities

##### Repositories

##### Domain Services

#### Application Layer

##### Application Services

##### Data Transfer Objects

#### Data Access

##### Entity Framework Core Integration

##### MongoDB Integration

### Customizing/Extending Modules

#### Overall

#### Module Entity Extension System

#### Customizing/Extending Entities

#### Customizing/Overriding Services

## Domain Driven Design

### Overall

### Domain Layer

#### Entities & Aggregate Roots

#### Value Objects

#### Repositories

#### Domain Services

#### Specifications

### Application Layer

#### Application Services

#### Data Transfer Objects

#### Unit Of Work

### E-Book: Implementing DDD

## Multi Tenancy

## Microservices

# API

## ABP Endpoints

### Application Configuration

### Application Localization

## API Versioning

## Auto API Controllers

## Dynamic C# API Clients

## Integration Services

## Static C# API Clients

## Swagger Integration

# User Interface

## MVC / Razor Pages

### Overall

### Navigation / Menus

### Forms & Validation

### Modals

### Data Tables

### Auto-Complete Select

### Page Alerts

### Dynamic JavaScript API Client Proxies

### Static JavaScript API Client Proxies

### Client Side Package Management

### Bundling & Minification

### Tag Helpers

#### Form Elements

#### Dynamic Forms

### Widgets

### Toolbars

### Page Header

### Branding

### Layout Hooks

### Testing

### Theming

#### The Basic Theme

#### LeptonX Lite

### JavaScript API

#### Overall

#### Localization

#### Auth

#### Current User

#### Settings

#### Features

#### AJAX

#### Message

#### Notify

#### Block/Busy

#### Events

#### DOM

#### Logging

#### Resource Loader

### Customize/Extend the UI

#### Overall

#### Entity Action Extensions

#### Data Table Column Extensions

#### Page Toolbar Extensions

### Security

#### Security Headers

## Blazor

### Overall

### Navigation / Menu

### Localization

### Theming

#### Overall

#### The Basic Theme

#### LeptonX Lite

#### Branding

#### Page Header

#### Page Layout

#### Toolbars

### Security

#### Authentication

#### Authorization

### Services

#### Current User

#### Current Tenant

#### Notification

#### Message

#### Page Alerts

#### Page Progress

### Other Components

#### SubmitButton

### Settings

### Error Handling

### Customization / Overriding Components

### Global Scripts & Styles

### Global Features

### Routing

### PWA Configuration

### Layout Hooks

## Angular

### Quick Start

### Development

#### Environment Variables

#### About Feature Libraries

#### Service Proxies

#### PWA Configuration

#### Unit Testing

### Core Functionality

#### Config State Service

#### Authorization

#### Current User

#### HTTP Requests

#### Localization

#### Form Validation

#### Settings

#### Features

#### Global Features

#### Permission Management

#### Multi Tenancy

#### Account Module

### Utilities

#### Managing RxJS Subscriptions

#### Working with Lists

#### Easy \*ngFor trackBy

#### Router Events

#### UI/Angular/Router-Events.md

#### Inserting Scripts & Styles to DOM

#### Lazy Loading Scripts & Styles

#### Projecting Angular Content

#### Modal

#### Confirmation Popup

#### Loading Directive

#### Toast Overlay

#### Page Alerts

#### Ellipsis

#### Context Strategy

#### Cross Origin Strategy

#### Dom Strategy

#### Container Strategy

#### Content Security Strategy

### Customization

#### Customization Guide

#### Theming

##### Overall

##### Configuration

##### The Basic Theme

##### LeptonX Lite

#### Modifying the Menu

#### Component Replacement

#### Extensions

##### Overall

##### Entity Action Extensions

##### Data Table Column Extensions

##### Page Toolbar Extensions

##### Dynamic Form Extensions

##### Date, time and datetime format pipes

### Components

#### Page

#### Chart

#### Card

## React Native

### Getting Started

## Common

#### Themes

#### Overriding the User Interface

#### Utilities

##### Linked List (Doubly)

# Data Access

## Overall

## Entity Framework Core

### Database Migrations

### Switch DBMS

#### To MySQL

#### To PostgreSQL

#### To Oracle

##### Oracle (Official)

##### Oracle (Devart)

#### To SQLite

## MongoDB

## Dapper

# Real Time

## SignalR Integration

# Dapr Integration

# Testing

# Deployment

## Configuring OpenIddict

## Configuring for Production

## Deploying to a Clustered Environment

## Distributed / Microservice Solutions

## Optimizing for Production

# Application Modules

## Overall

## Account

## Audit Logging

## Background Jobs

## CMS Kit

## Docs

## Feature Management

## Identity

## IdentityServer

### IdentityServer Migration Guide

## OpenIddict

### OpenIddict Migration Guide

## Permission Management

## Setting Management

## Tenant Management

## Virtual File Explorer

## Common

### Database Tables

# Samples

## All Samples

## eShopOnAbp

## EventHub

## Microservice Demo (legacy)

# Books

## Mastering ABP Framework

## Implementing Domain Driven Design

# Release Information

## Upgrading

## Official Packages

## Preview Releases

## Nightly Builds

## Road Map

## Migration Guides

# Contribution Guide