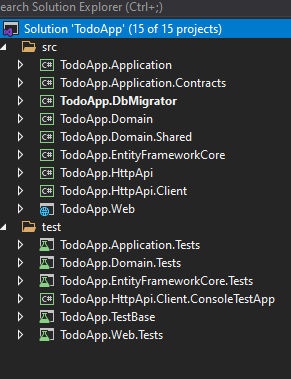
ABP Framework

Open source framework based on .net core.

It handles basic features of the web site like authentication, authorization, localization,…

* Creating project:

1. Using CLI
2. Download from website

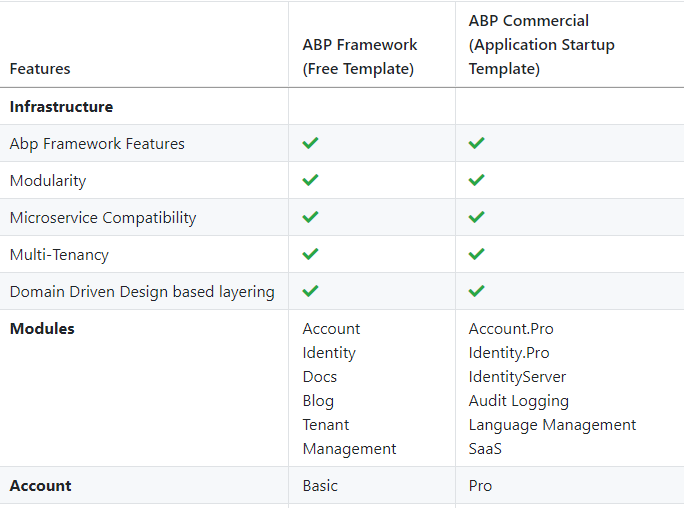


Project is implemented based on DDD

* Framework structure

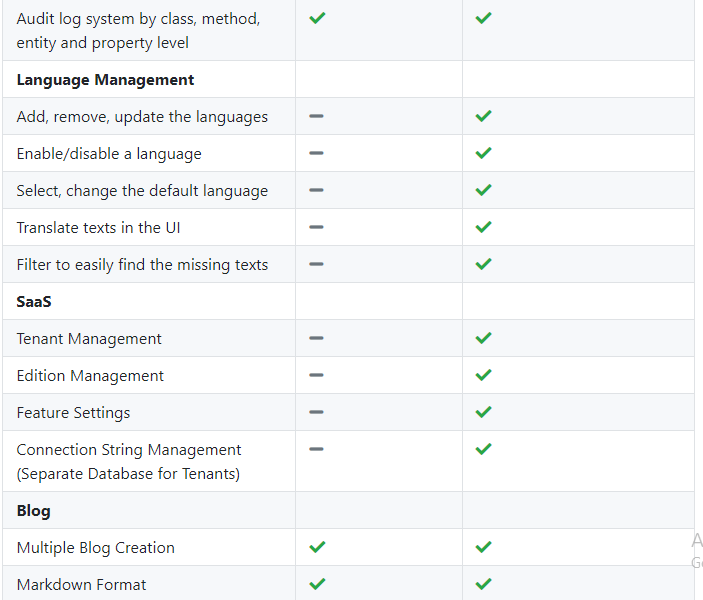
1. Domain layer is separated into two projects:
   1. Domain layer contain repositories/ entities/ interfaces
   2. Domain shared : shared between all layers contains constant and enum values
2. Application layer is separated into two layers:
   1. Contracts: contains interface to application services and DTO
   2. Application project: implementation of services in contracts
3. Framework core: db context and migration to unified db migration code first
4. Db migrator: console application, simplifies migrations and seeds the db
5. http api: api controllers
6. http api client project client proxies which uses http api projects, that can be shared to consume api
7. web project: UI and has the connection string

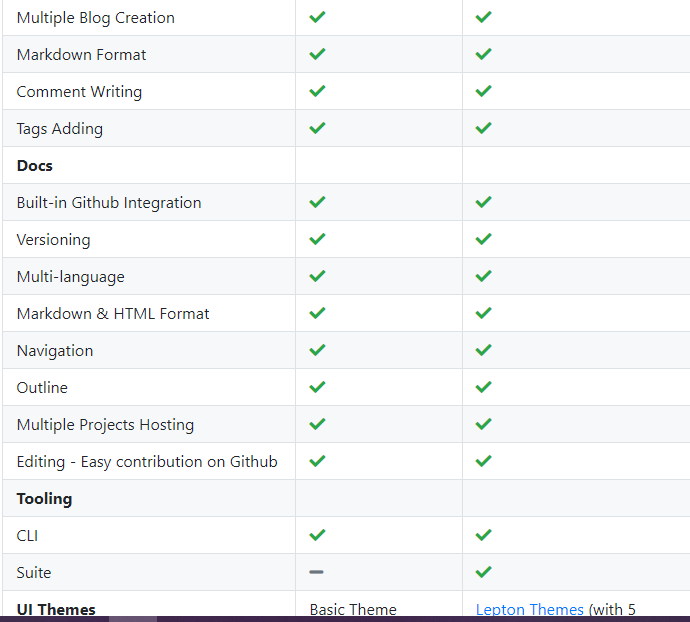
* ABP Framework vs ABP Commercial









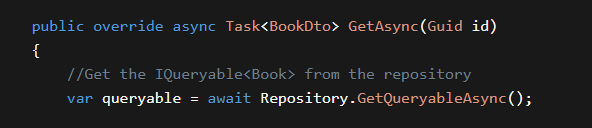


* Framework features:

1. Login page
2. Localization
3. Menu
4. Roles management and permissions
5. Profile page
6. Base classes for entities
   1. Audit
   2. Aggregate & Aggregate Root(DDD Design)
   3. Convention method gracefully configures/maps the inherited properties. Always use it for all your entities

Note:

* To customize api functions make overload for it



* Run migrator to create db; then run ui

Built in app is using identity, talent management (role and permission) ,..

* **Steps to create project:**

<https://docs.abp.io/en/abp/latest/Tutorials/Part-1?UI=MVC&DB=EF>

1. Install abp framework

dotnet tool install -g Volo.Abp.Cli

1. Create a new project using cli command to download the project template

abp new TodoApp(project name)

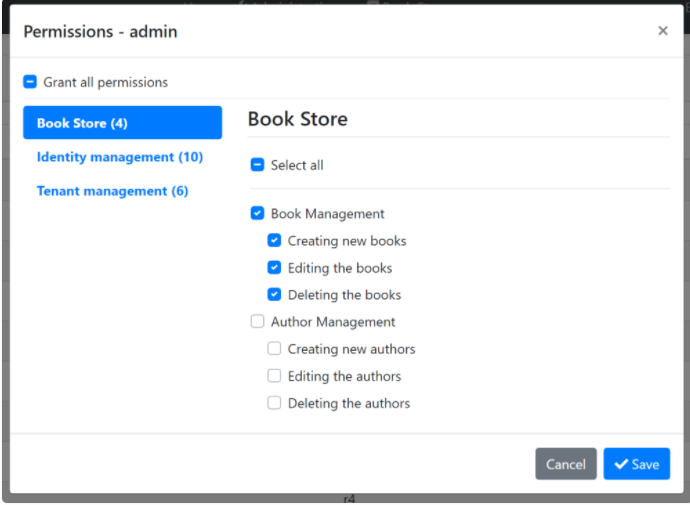
1. On TodoApp.DbMigrator
   1. R-Click to run to create the db and seeds the data
2. On TodoApp.Web
   1. R-Click to run UI
   2. use admin as the username and 1q2w3E\*
3. TodoApp.Domain
   1. Create a folder->books->entity book inherits from audit aggregate root and aggregate root for DDD(aggregate root means the base class aggregate means the derived or the child class)
4. TodoApp.Domain.Shared
   1. Create the enum
5. TodoApp.EntityFrameworkCore
   1. Map the book entity to db by creating db set
   2. Add the entity inside OnModelCreating to create table
   3. Add Database Migration
      1. Make ui as start up project
      2. Open package console and write

add-migration Created\_Book\_Entity

or open the directory ->dotnet ef migrations add Created\_Book\_Entity

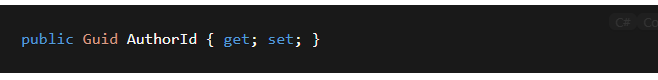
* + 1. Add data inside the table
       1. Create folder SeedData
       2. Create class BookStoreDataSeederContributor
    2. Update current db
       1. Run migrator or in console update-database
  1. Creating service
     1. Create BookDto
     2. Adjust auto mapper inside application project for the entity TodoAppApplicationAutoMapperProfile add mapping
     3. Create dto for each CRUD operation it’s free and simple
     4. Create the interface IBookAppService
     5. Implement the service BookAppService inside application project
  2. Expose service api end point, ABP can [**automagically**](https://docs.abp.io/en/abp/latest/API/Auto-API-Controllers) configures your application services as MVC API Controllers by convention.
     1. Run ui application <https://localhost:44342/swagger/index.html>
     2. Testing in the Developer Console, ABP **dynamically** creates [JavaScript Proxies](https://docs.abp.io/en/abp/latest/UI/AspNetCore/Dynamic-JavaScript-Proxies) for all API endpoints. So, you can use any **endpoint** just like calling a **JavaScript function**.
        1. Open the console and write   
           todoApp.services.book.getList({}).done(function (result) { console.log(result); });
  3. Adjust localization inside TodoApp.Domain.Shared , We prefer some conventions for specific text types;
     1. Add Menu: prefix for menu items.
     2. Use Enum:<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.
  4. Dealing With UI inside TodoApp.Web
     1. Create folder Books -> Index
     2. Add book inside menu TodoApp.Web.Menus-> TodoAppMenuContributor
     3. Adding mapping

CreateMap<CreateUpdateBookDto, Book>();

1. Inside TodoApp.Application.Tests Creating Unit Test
   1. Open the **Test Explorer Window** (use Test -> Windows -> Test Explorer menu if it is not visible) and **Run All** tests:
2. Permissions

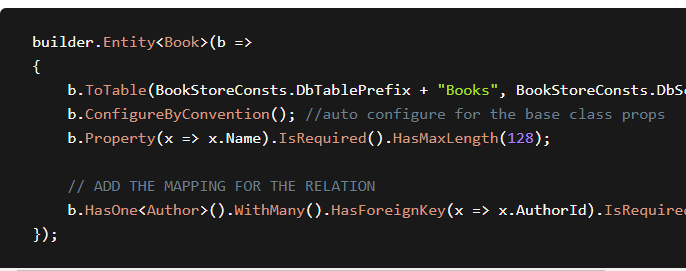
ABP Framework provides an [authorization system](https://docs.abp.io/en/abp/latest/Authorization) based on the ASP.NET Core's [authorization infrastructure](https://docs.microsoft.com/en-us/aspnet/core/security/authorization/introduction). One major feature added on top of the standard authorization 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.

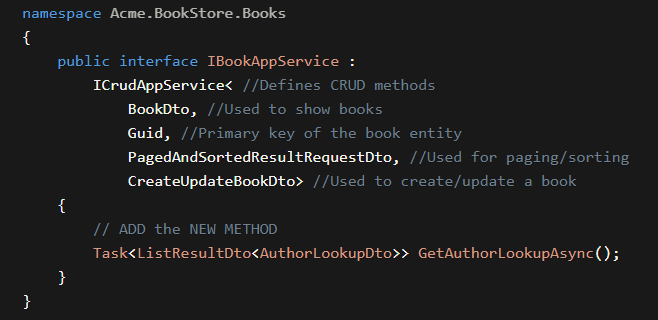
* Define them inside Contracts project-> TodoAppPermissions class
* Register them inside BookStorePermissionDefinitionProvider
* Apply them
  + Secure API inside services inside TodoApp.Application
  + Secure Front-End inside TodoApp.Web-> TodoAppWebModule-> ConfigureServices While securing the HTTP 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.
    - Adjust the page buttons too
    - Adjust menu items
* **Relation between book and author:**
  + - In book entity add

Add relation by id

This is due to follow the DDD best practices (rule: refer to other aggregates only by id).

* + - Update db using migration
* **Getting Look-Up data from entity**

Open the IBookAppService interface in the Books folder of the Acme.BookStore.Application.Contracts project and add a new method, named GetAuthorLookupAsync,



Author:

1. Domain
   * + - Author class
       - Author Manager
       - Validation class

## IAuthorRepository

* + - * Seed Data

1. EntityFrameworkCore
   1. Add data set
   2. Create migration
   3. EfCoreAuthorRepository-> implement the interface
2. Contract
   1. Create interface & dto
   2. Add permission
3. Application
   1. Auto-mapper
4. **Migrate**
   1. **Run migration**
5. Tests

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.

# **Exception Handling**

[Exception Handling | Documentation Center | ABP.IO](https://docs.abp.io/en/abp/latest/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.

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).

### 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.

## Send exception details to the client

You can send exceptions to the client via the SendExceptionsDetailsToClients property of the AbpExceptionHandlingOptions class:

* Note:

1. **Implement DataTable**
2. **(Auto-Save):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.

\*Note:

Make Service name like its interface as it maps by Name