

INDOTALENT



Project Structure

Introduction

This document will introduce the structure of the project. The project itself use and follow the ABP Framework product and documentation. ABP Framework created by Volosoft

ABP Framework is a complete infrastructure based on the ASP.NET Core to create modern web applications and APIs by following the software development best practices and the latest technologies.

ABP Framework is free and open source.

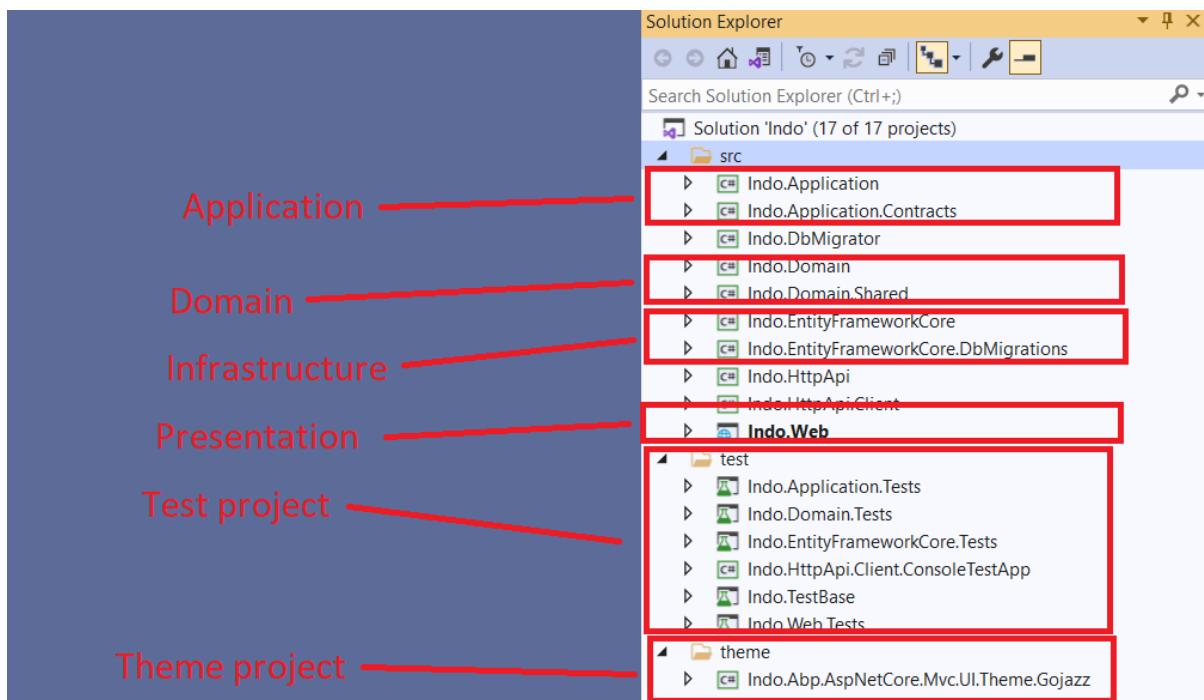
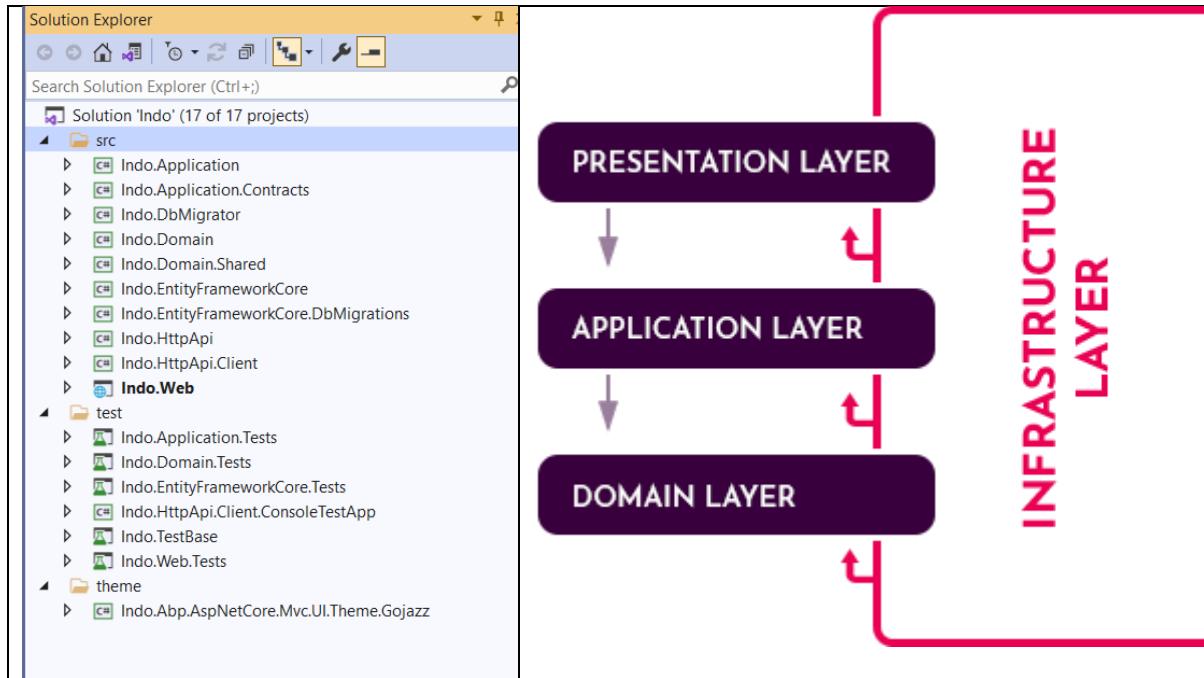
More about ABP Framework can be read on their official sites:

- Documentation: <https://docs.abp.io/en/abp/latest>
- Project Source Code: <https://github.com/abpframework/abp>
- Official Volosoft Company Website: <https://volosoft.com/>

ASP.NET Core 5 Razor Pages

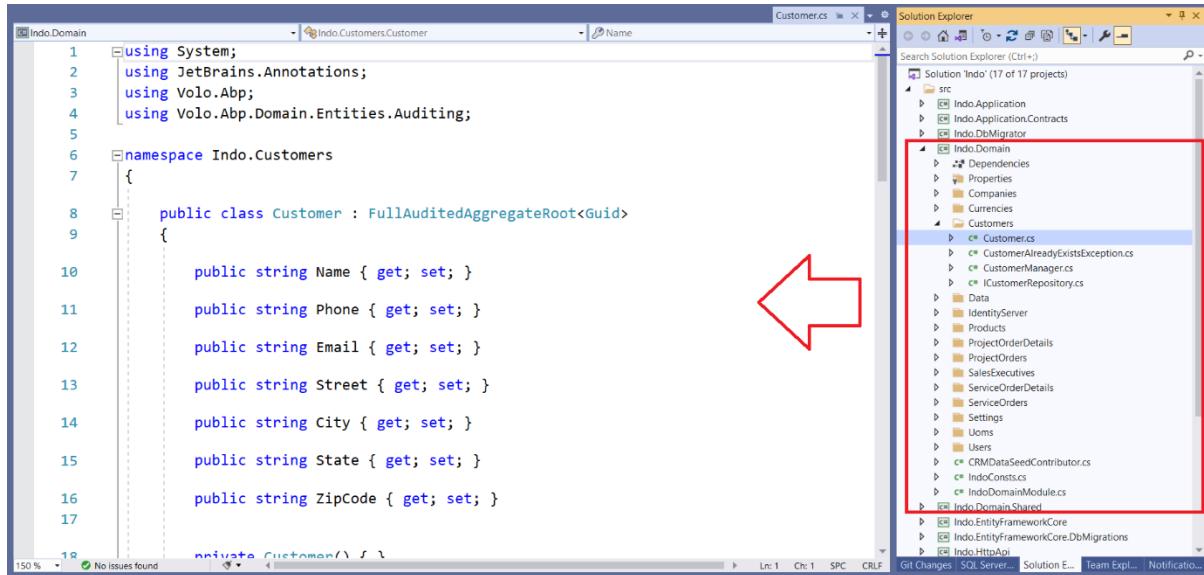
ABP Framework can be implemented using various Web UI technologies such as: Angular, Blazor, MVC and Razor Pages.

This product designed specifically to be used on ASP.NET Core 5 Razor Pages. Below is the complete project structure that follow the clean architecture:



Domain Layer

Includes business objects and the core (domain) business rules. This is the heart of the application.



A screenshot of the Visual Studio IDE showing the Domain layer structure. The left pane shows the code editor for `Customer.cs` in the `Indo.Domain` project. The right pane shows the Solution Explorer with the `Indo.Domain` project selected. A red arrow points from the Solution Explorer back to the code editor, indicating the relationship between the domain objects and their implementation.

```
1 using System;
2 using JetBrains.Annotations;
3 using Volo.Abp;
4 using Volo.Abp.Domain.Entities.Auditing;
5
6 namespace Indo.Customers
7 {
8     public class Customer : FullAuditedAggregateRoot<Guid>
9     {
10         public string Name { get; set; }
11         public string Phone { get; set; }
12         public string Email { get; set; }
13         public string Street { get; set; }
14         public string City { get; set; }
15         public string State { get; set; }
16         public string ZipCode { get; set; }
17
18     private Customer() { }
19 }
```

Solution Explorer:

- Indo (17 of 17 projects)
 - src
 - Indo.Application
 - Indo.Application.Contracts
 - Indo.DbMigrator
 - Indo.Domain
 - Dependencies
 - Properties
 - Companies
 - Currencies
 - Customers
 - Customer.cs
 - CustomerAlreadyExistsException.cs
 - CustomerManager.cs
 - ICustomerRepository.cs
 - Data
 - IdentityServer
 - Products
 - ProjectOrderDetails
 - ProjectOrders
 - SalesExecutives
 - ServiceOrderDetails
 - ServiceOrders
 - Settings
 - Uoms
 - Users
 - CRMDataSeedContributor.cs
 - IndoConsts.cs
 - IndoDomainModule.cs
 - Indo.Domain.Shared
 - Indo.EntityFrameworkCoreCore
 - Indo.EntityFrameworkCoreCore.DbMigrations
 - Indo.HttpApi

Application Layer

Mediates between the presentation and domain layers. Orchestrates business objects to perform specific application tasks. Implements use cases as the application logic.

The screenshot shows the Visual Studio IDE interface. On the left is the code editor window titled 'CustomerAppService.cs' with the following C# code:

```
21 public CustomerAppService(
22     ICustomerRepository customerRepository,
23     CustomerManager customerManager,
24     IProjectOrderRepository projectOrderRepository,
25     IServiceOrderRepository serviceOrderRepository
26     )
27     {
28         _customerRepository = customerRepository;
29         _customerManager = customerManager;
30         _projectOrderRepository = projectOrderRepository;
31         _serviceOrderRepository = serviceOrderRepository;
32     }
33     public async Task<CustomerDto> GetAsync(Guid id)
34     {
35         var obj = await _customerRepository.GetAsync(id);
36         return ObjectMapper.Map<Customer, CustomerDto>(obj);
37     }
38     public async Task<PagedResultDto<CustomerDto>> GetListAsync(PagedAndSortedResu
39     {
40         var queryable = await _customerRepository.GetQueryableAsync();
41         var query = from customer in queryable
42                     select new { customer };
43         queryable = queryable

```

The Solution Explorer window on the right lists various projects and files under the solution 'Indo'. A red box highlights the 'Customers' folder, which contains the 'CustomerAppService.cs' file. A red arrow points from the code editor towards this highlighted folder.

Presentation Layer

Provides an interface to the user. Uses the application layer to achieve user interactions.

The screenshot shows the Visual Studio IDE interface. On the left is the Solution Explorer, which lists the project structure for 'Indo.Web'. A red box highlights the 'Pages' folder under 'Indo.Web', and a red arrow points from this box to the code editor on the right. The code editor displays the file 'Edit.cshtml.cs' containing C# code for a controller action. The code includes methods for handling GET and POST requests, interacting with an 'ICustomerAppService' to get and update customer data, and mapping between DTOs and view models using 'ObjectMapper'. The code editor has syntax highlighting and line numbers. The status bar at the bottom shows '150 %' zoom, line 1, character 1, and other standard status indicators.

```
17 public EditModel(ICustomerAppService customerAppService)
18 {
19     _customerAppService = customerAppService;
20 }
21 public async Task OnGetAsync(Guid id)
22 {
23     var dto = await _customerAppService.GetAsync(id);
24     Customer = ObjectMapper.Map<CustomerDto, EditCustomerViewModel>(dto);
25 }
26 public async Task<IActionResult> OnPostAsync()
27 {
28     try
29     {
30         await _customerAppService.UpdateAsync(
31             Customer.Id,
32             ObjectMapper.Map<EditCustomerViewModel, UpdateCustomerDto>(Customer));
33     }
34     return NoContent();
35 }
36 catch (CustomerAlreadyExistsException ex)
37 {
38     throw new UserFriendlyException($"{ex.Code}");
39 }
```

Menu Management

Provides mechanism to manage the menus.

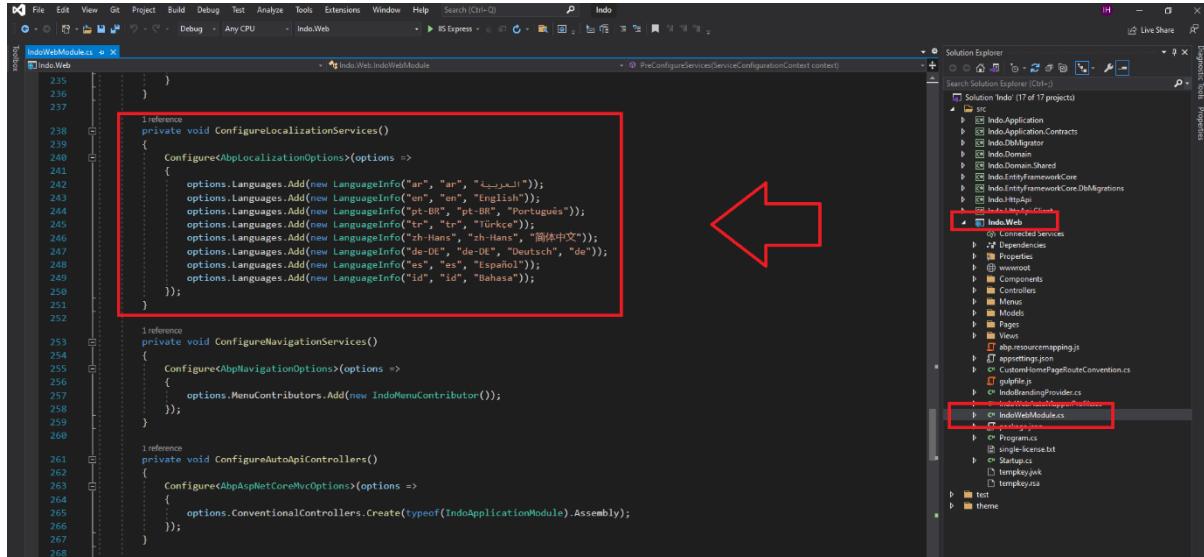
RBAC Management

RBAC or Roles Based Access Control, before can be used at user interface level, its definition should be created first. The source code located at Application Contracts project.

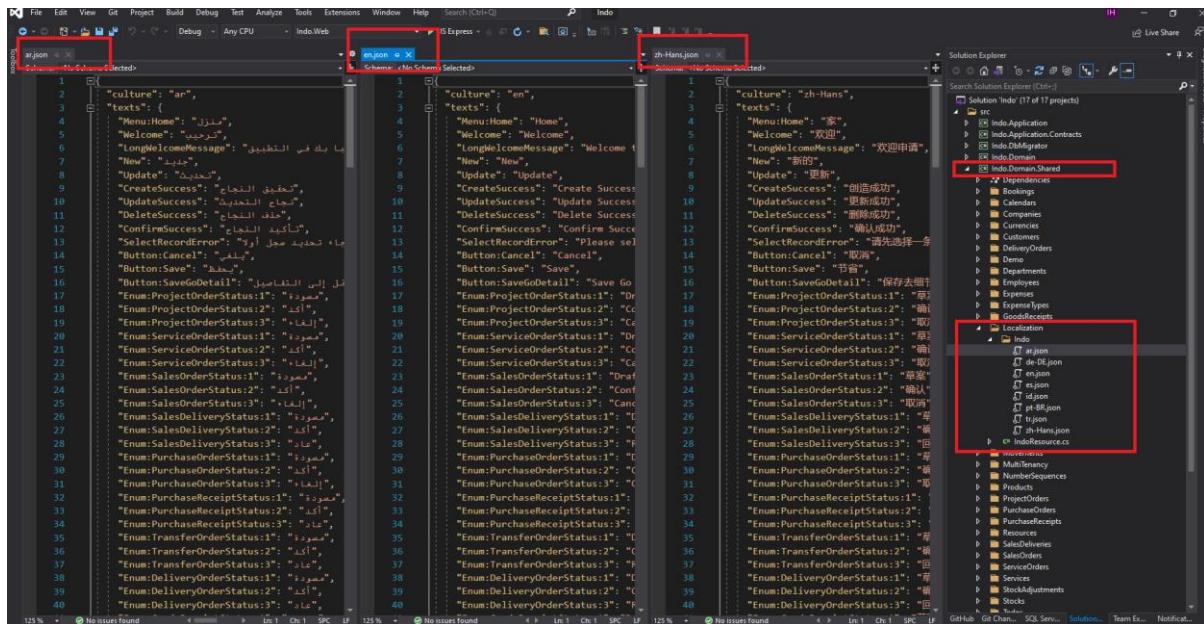
Multi Language Management

Localization for multi language can be achieved easily by configuring each language with its translations in the format of JSON.

Language that will be supported, can be configure at Web project:



The translations, can be configured at Domain Shared project:



User Interface (UI)

The UI following Microsoft official ASP.NET Core Razor Pages approach, which is the desired and recommended approach by Microsoft compare to classic MVC approach.

All the UI resides under the Pages folder of Web project and grouped by folders which contains CSHTML, CSS and JavaScript.

The screenshot shows a Visual Studio interface with several windows open:

- Left pane:** Shows the file structure of the project. The **Pages** folder is highlighted with a red box, containing files like `Booking.cshtml`, `CalendarView.cshtml`, `Create.cshtml`, `Help.cshtml`, `Index.cshtml`, and `Update.cshtml`.
- Middle pane:** Contains three code editors:
 - Index.js:** A file using the `import` statement to bring in various modules.
 - Index.css:** A CSS file defining styles for a modal dialog.
 - Create.cshtml:** An ASP.NET Core view page with a form and several input fields.
- Right pane:** The **Solution Explorer** window, which lists all the projects and files in the solution, including `Indo`, `Indo.Application`, `Indo.Contracts`, `Indo.Deliverer`, `Indo.Localization`, `Indo.Localization.Shared`, `Indo.EntityFrameworkCore`, `Indo.EntityFrameworkCore.DMigrations`, `Indo.HttpApi`, and `Indo.Web`.

REST API

All the method at Application project, will be auto magically converted into REST API

The screenshot shows the Visual Studio IDE with the CustomerAppService.cs file open in the code editor. The file contains C# code for a service class that interacts with various repositories. The Solution Explorer on the right shows multiple projects under the 'src' folder, with a red box highlighting the 'CustomerAppService.cs' file.

```
16  i reference
17  public class CustomerAppService : IndoAppService, ICustomerAppService
18  {
19      private readonly ICustomerRepository _customerRepository;
20      private readonly CustomerManager _customerManager;
21      private readonly IProjectOrderRepository _projectOrderRepository;
22      private readonly IServiceOrderRepository _serviceOrderRepository;
23      private readonly ISalesOrderRepository _salesOrderRepository;
24
25      public CustomerAppService(
26          ICustomerRepository customerRepository,
27          CustomerManager customerManager,
28          IProjectOrderRepository projectOrderRepository,
29          IServiceOrderRepository serviceOrderRepository,
30          ISalesOrderRepository salesOrderRepository
31      )
32      {
33          _customerRepository = customerRepository;
34          _customerManager = customerManager;
35          _projectOrderRepository = projectOrderRepository;
36          _serviceOrderRepository = serviceOrderRepository;
37          _salesOrderRepository = salesOrderRepository;
38      }
39
40      public async Task<CustomerReadDto> GetAsync(Guid id)
41      {
42          var obj = await _customerRepository.GetAsync(id);
43          return ObjectMapper.Map<Customer, CustomerReadDto>(obj);
44      }
45
46      public async Task<List<CustomerReadDto>> GetListAsync()
47      {
48          var queryable = await _customerRepository.GetQueryableAsync();
49          var query = from customer in queryable
50                      select new { customer };
51          var queryResult = await AsyncExecutor.ToListAsync(query);
52          var dtos = queryResult.Select(x =>
53          {
54              var dto = ObjectMapper.Map<Customer, CustomerReadDto>(x.customer);
55              return dto;
56          }).ToList();
57      }
58  }
```

Check and test using swagger (already pre configured, no additional steps required)

Using URL: <https://localhost:xxx/swagger>

The screenshot shows the Swagger UI interface for the Customer endpoint. It lists several operations: GET /api/app/customer/{id} (blue button), PUT /api/app/customer/{id} (orange button), DELETE /api/app/customer/{id} (red button), GET /api/app/customer (light blue button), and POST /api/app/customer (green button). The 'Customer' section is expanded, showing these specific endpoints.

