**Apply security, Ajax request with RESTful API and Web Application**

# Introduction

Imagine you're an employee of a product retailer named Product Store. Your manager has asked you to develop an application for simple product management. The relationship between Category and Product is One-to-Many, one product is belong to only one Category, one category will have zero or many products. The Product includes these properties: ProductId, ProductName, Category, UnitsInStock, UnitPrice. The Category includes properties: such as CategoryId, CategoryName. The application has to support adding, viewing, modifying, and removing products - a standardized usage action verbs better known as Create, Read, Update, Delete (CRUD).

This lab explores creating an application using ASP.NET Core Web API to create RESTful API, and ASP.NET Core Web App MVC and Identity and AJAX. An **SQL Server** **Database** will be created to persist the product data that will be used for reading and managing product data by **Entity Framework Core.**

# Lab Objectives

In this lab, you will:

* Use the Visual Studio.NET to create ASP.NET Core Web Web API Project.
* Develop Web application using MVC Pattern combination with Identity and AJAX techniques.
* Use Entity Framework Core to create a SQL Server database (Forward Engineering Approach).
* Develop Entity classes, DBContext class, DAO class to perform CRUD actions using Entity Framework Core.
* Apply Repository pattern to develop application.
* Run the project and test the application actions.

# Guidelines

# Activity 01: Create a Blank Solution

**Step 01**. Create a Solution named **Lab03\_IdetityAjax ASP.NETCoreWebAPI**.

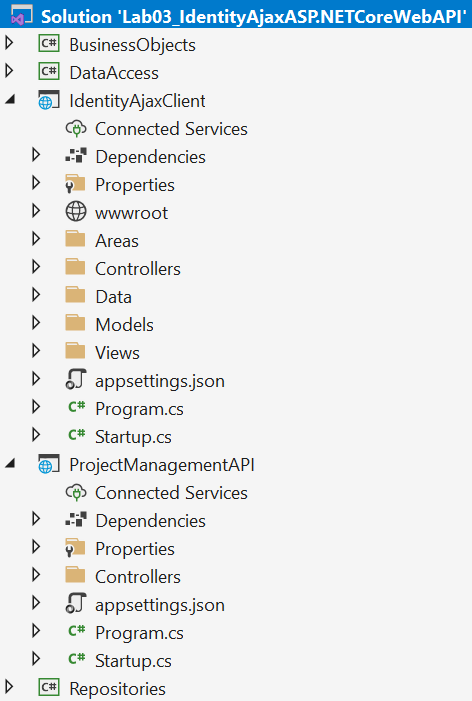
**Step 02**. Create Class Library Project: BusinessObjects.

**Step 03**. Create Class Library Project: Repositories.

**Step 04**. Create Class Library Project: DataAccess.

**Step 05**. Create ASP.NET Core Web Web API Project.

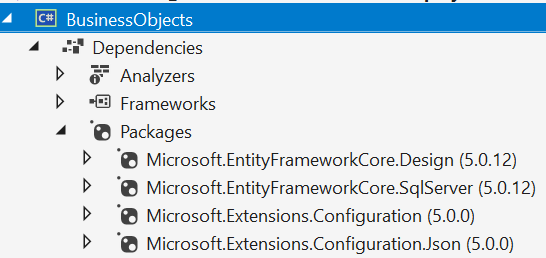
**Step 06**. Create ASP.NET Core Web Application (Model-View-Controller) Project using Security with Identity.



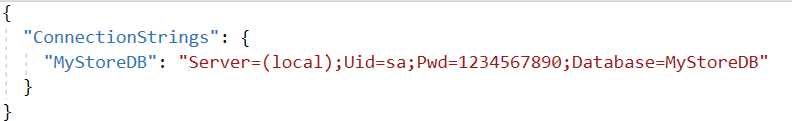
# Activity 02: BusinessObjects Project - Work with Entity Framework

**Step 01**. Create Class Library Project named BusinessObjects

**Step 02**. Install the following packages from NuGet:



**Step 03**. Add Connection string (also add JSON appsettings.json file)



<ItemGroup>

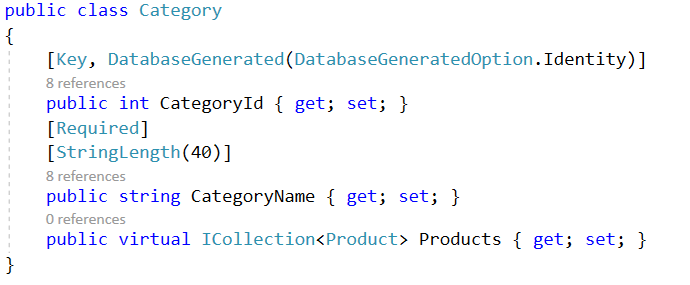
<None Update="appsettings.json">

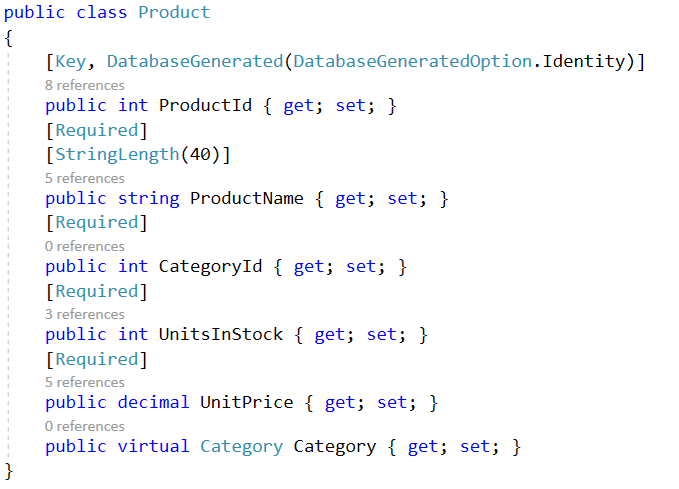
<CopyToOutputDirectory>Always</CopyToOutputDirectory>

</None>

</ItemGroup>

**Step 04**. Add “Products.cs”, “Category.cs” entities, and the context class “ApplicationDBContext.cs”







**Step 05**. Add-Migration and Update-Database

*dotnet ef migrations add “InitialDB”*

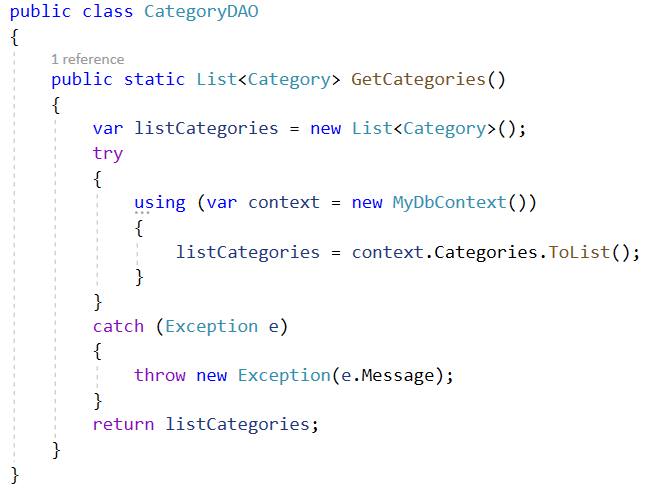
*dotnet ef database update*

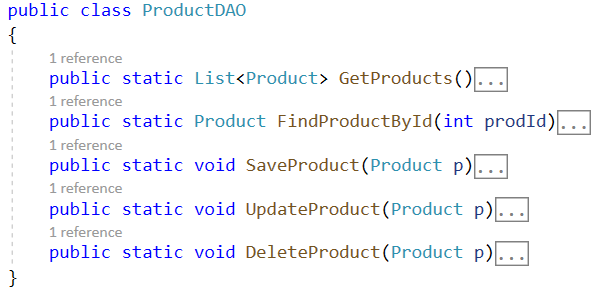
# Activity 03: DataAccess Project - contain methods for accessing the underlying database

**Step 01**. Create Class Library Project named DataAccess

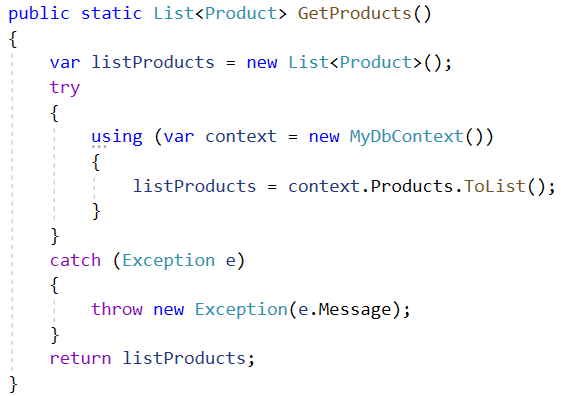
**Step 02**. Add Project reference: BusinessObjects Project

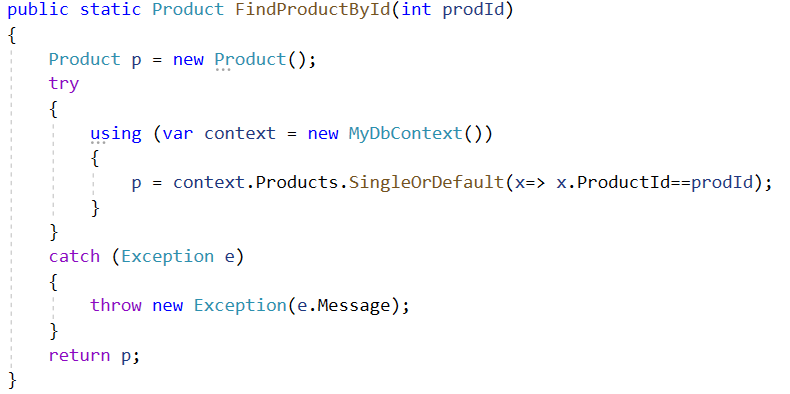
**Step 03.** Add data access classes for Product and Category

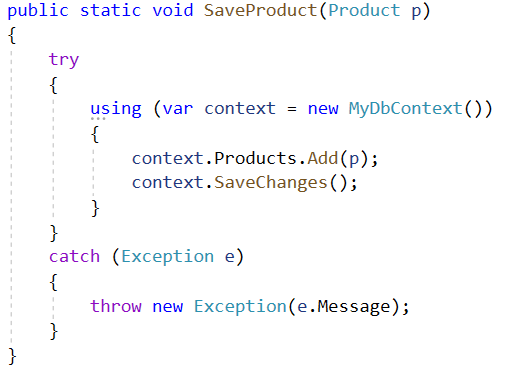




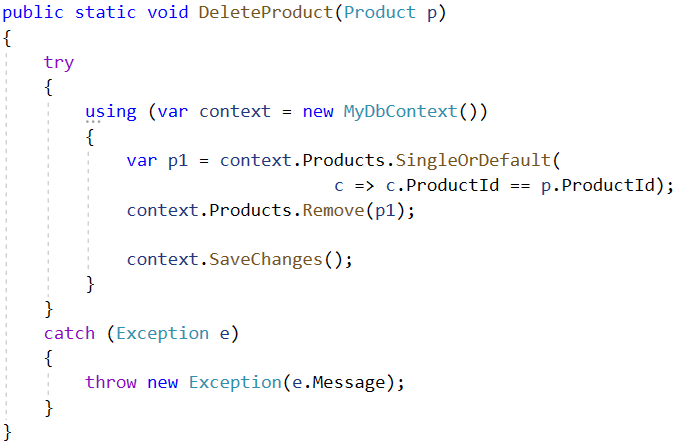
The detail functions for ProductDAO.cs









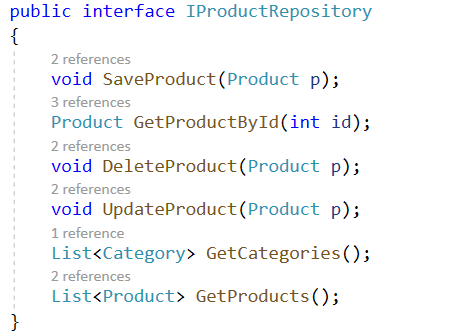


# Activity 04: Class Library Repositories Project - create an abstraction layer between the Data Access Layer and the Business Logic Layer of the application

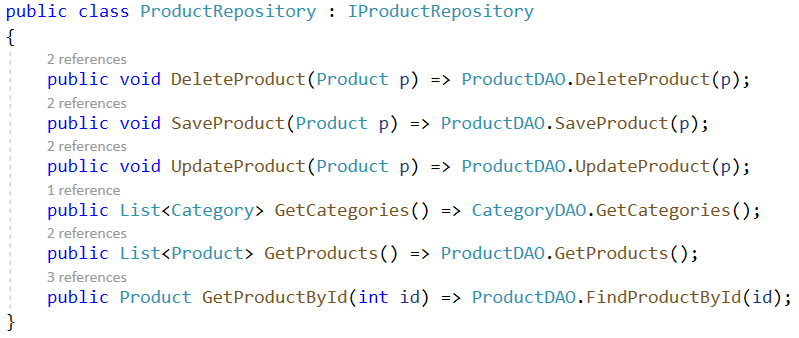
**Step 01**. Create Class Library Project named Repositories

**Step 02**. Add Project reference: BusinessObjects, DataAccess Projects

**Step 03**. Create IProductRepository Interface



**Step 04**. Create ProductRepository class implements IProductRepository Interface

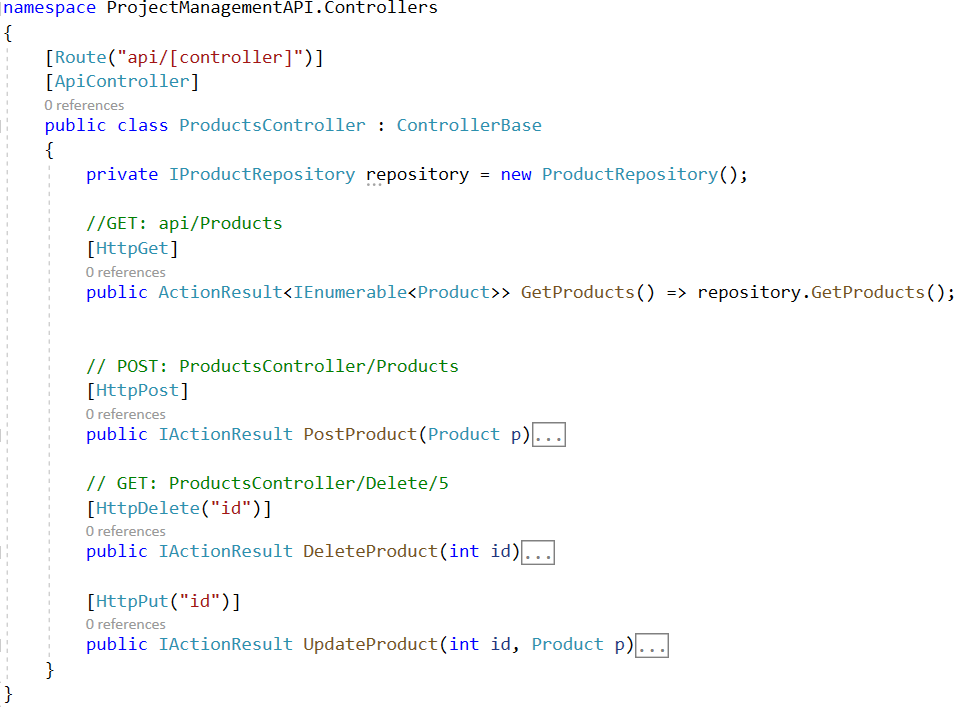


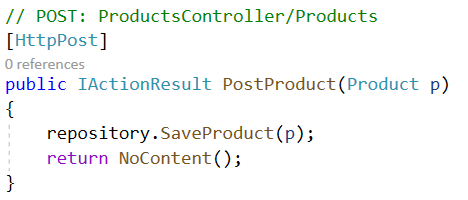
# Activity 05: Create ProductManagementAPI Project (Work with ASP.NET Core Web API template)

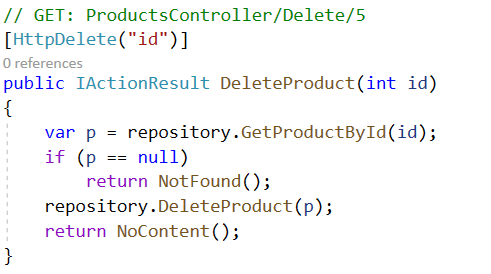
**Step 01**. Create ASP.NET Core Web API Project named ProductManagementAPI

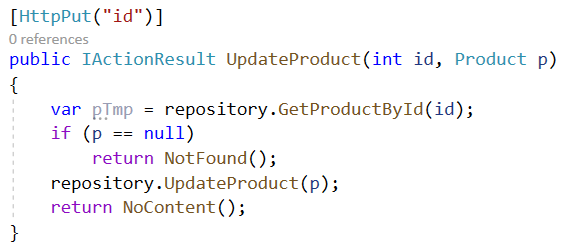
**Step 02**. Add Project reference: Repository Project

**Step 03**. Add ApiController named ProductsControllers.cs

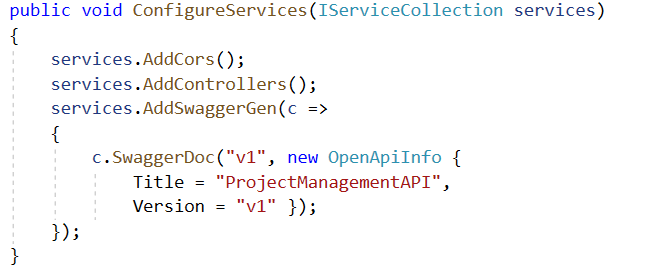


The detail of functions



  
**Step 04**. Create Web API Settings to allow Ajax request with Startup.cs

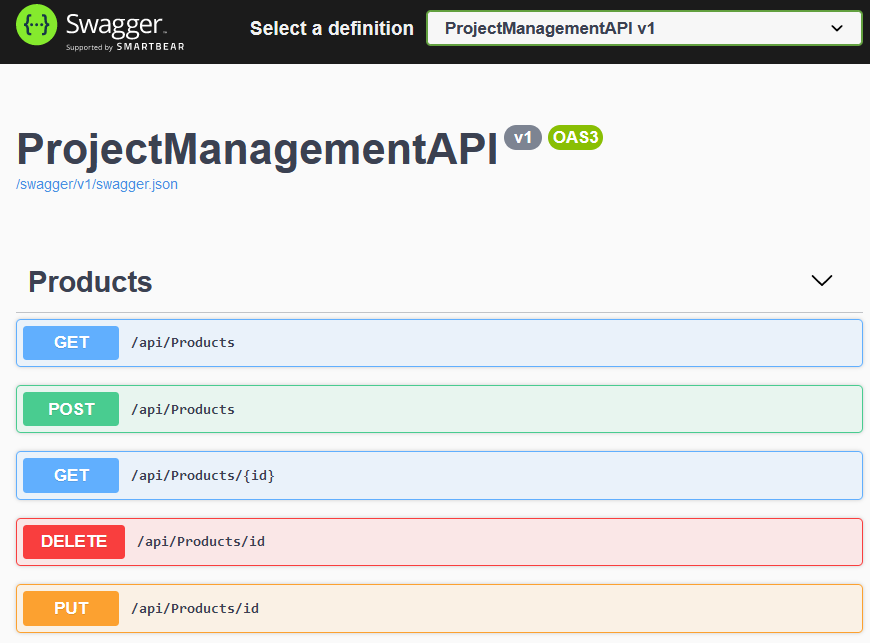
Configure with ConfigureServices() function



Configure with Configure () function

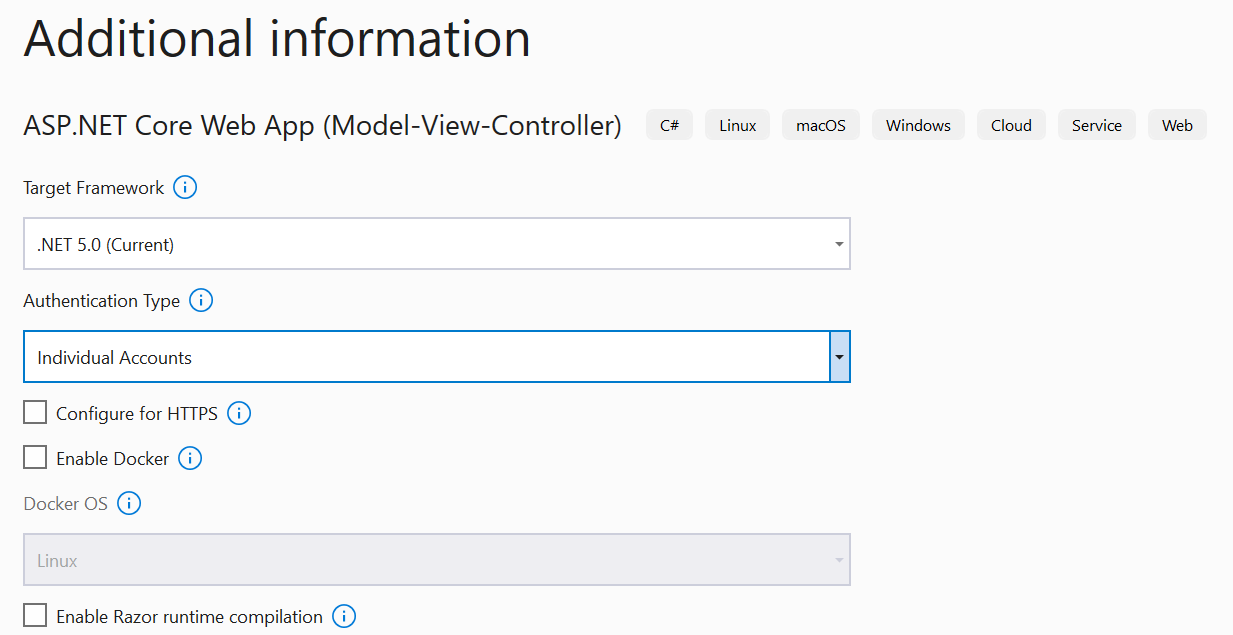
****

**Step 05**. Test API project with OpenAPI or Postman



# Activity 06: Implement Identity and Ajax in ASP.NET Core Web Application with Model-View-Controller Project

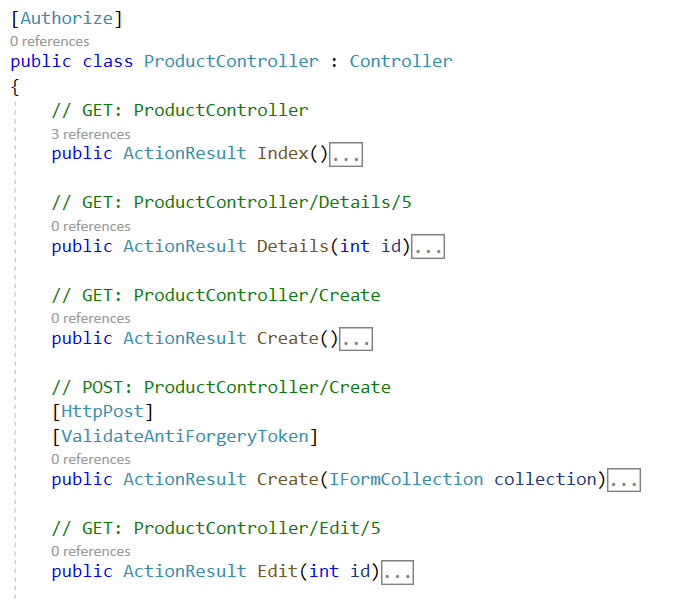
**Step 01**. Create ASP.NET Core Web App (Model-View-Controller) named IdentityAjaxClient

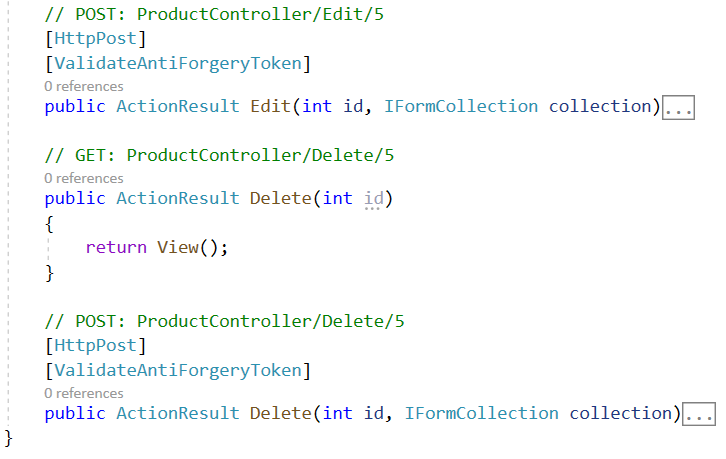


**Step 02**. Add Project reference: BusinessObjects Project (or create new DTO classes)

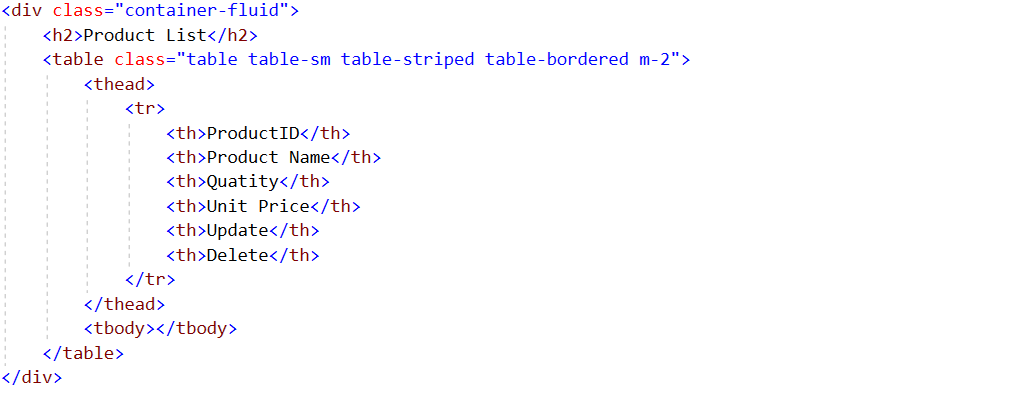
**Step 03**. Create Controller to connect to ProductManagementAPI

Simple create ProductController with [Authorize] and action methods return View. All actions will work with Ajax requests in View





**Step 04**. Create View with Ajax



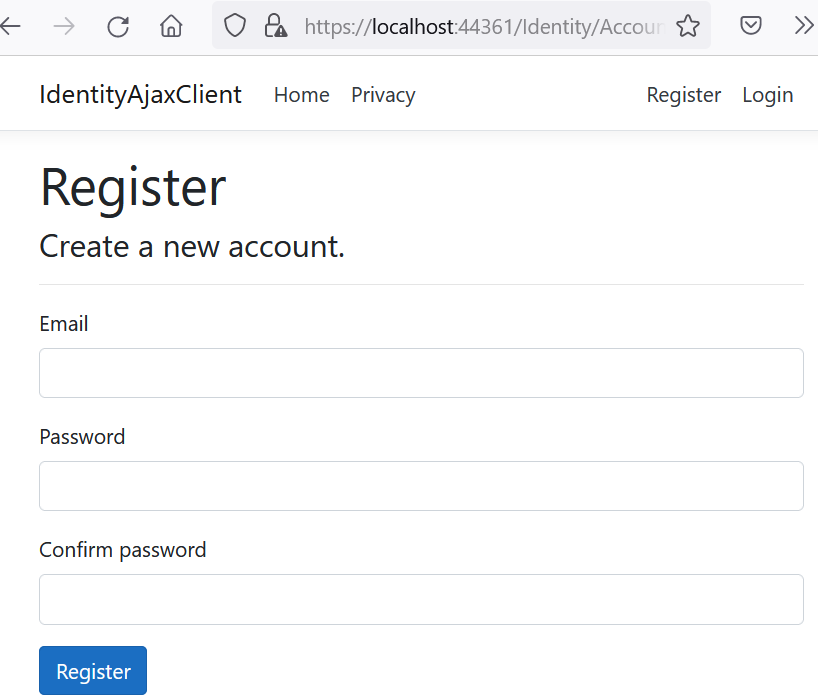


URL get from ASP.NET Core Web API

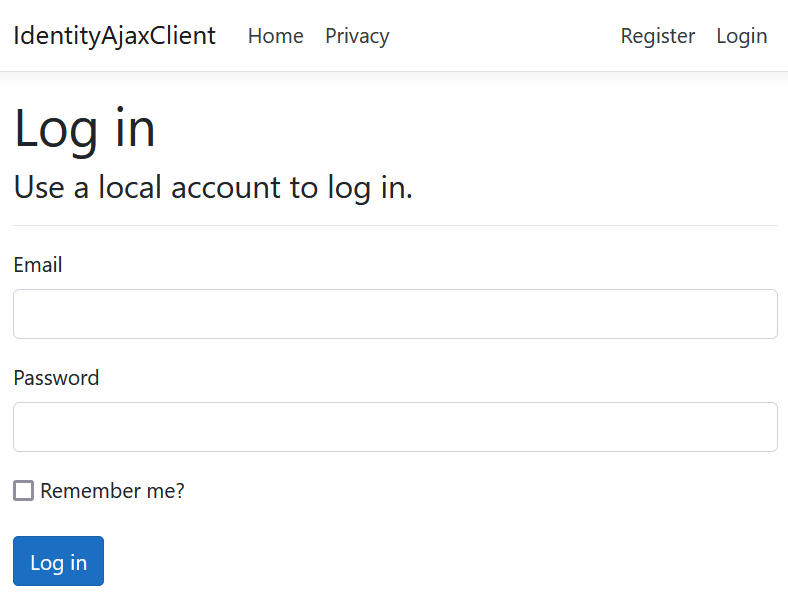


**Step 05**. Test the function of Web Client

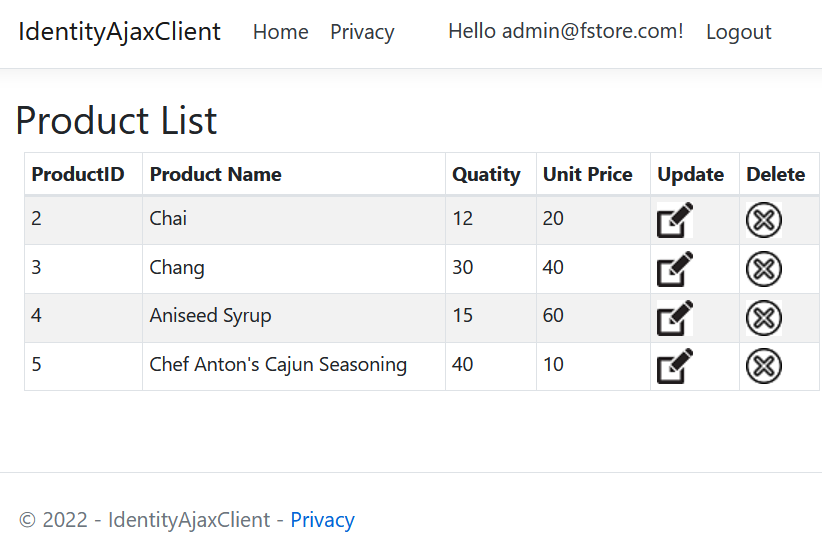
Registration UI



Login UI



List Product UI



# Activity 07: Build and run Project. Test all CRUD actions

Note: Choose the option for multiple startup projects.

