**Lab 01. Working with ASP.NET Core Web API**

# 1. Introduction

Imagine you're an employee of a store named **ProductStore**. Your manager has asked you to develop a WPF application for product management. 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 App (Model-View-Controller). An **SQL Server** **Database** will be created to persist the car's data that will be used for reading and managing product data by **Entity Framework Core**.

# 2. Lab Objectives

In this lab, you will:

* Use the Visual Studio.NET to create ASP.NET Core Web API and Class Library (.dll) projects.
* Create a SQL Server database named MyStoreDB that has a Product, Category, AccountMember tables.
* Apply Repository pattern in a project.
* Add CRUD action methods to ASP.NET Core Web API Controller.
* Run the project and test the application actions.

# 3. Database Design (MyStore)

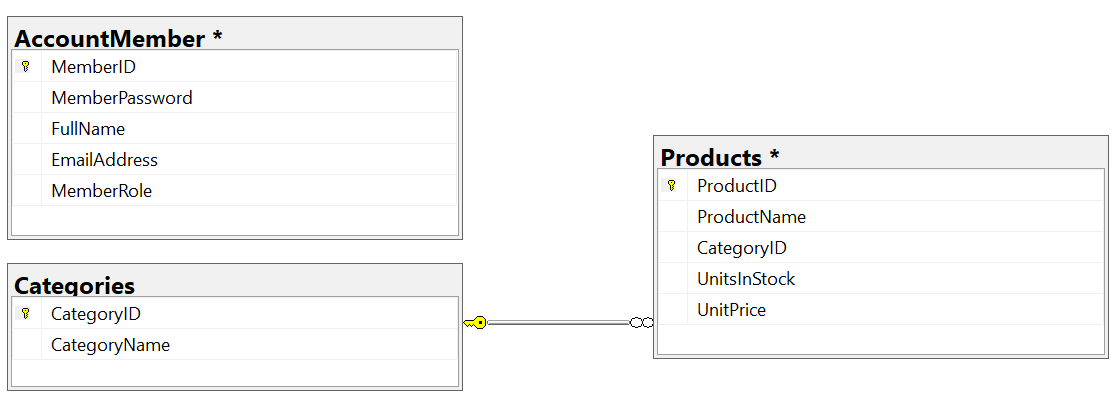


Table AccountMember

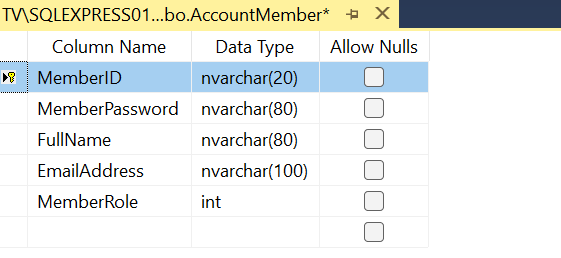


Table Categories

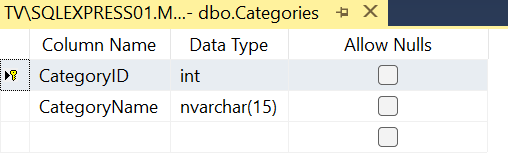
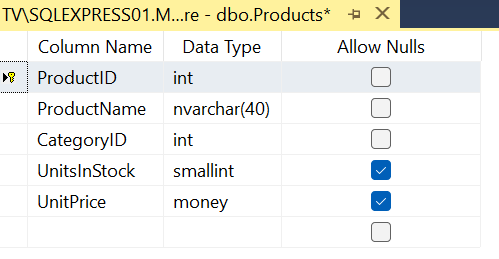


Table Products



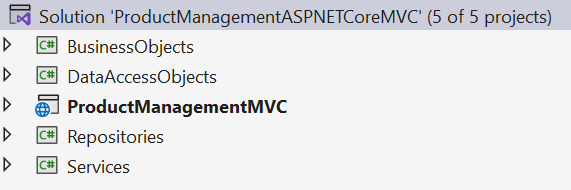
# Activity 01: Build a solution by Visual Studio.NET

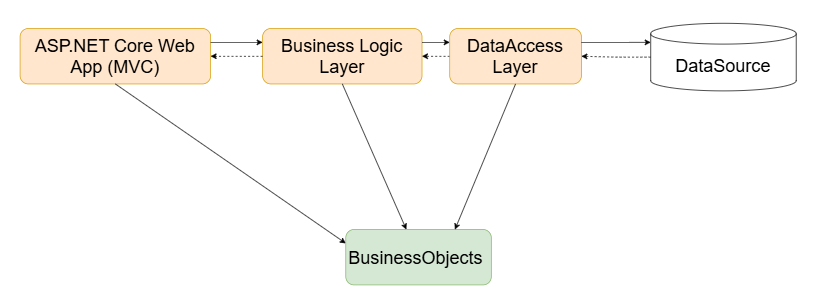
Create a Blank Solution named **ProductManagementASPNETCoreMVC** then add new a **Class Library** project named **BusinessObjects, DataAccessObjects, Repositories, Services** and a ASP.NET Core Web App (MVC) project named **ProductManagementMVC**

## **Step 01**. Create a Blank solution.

## **Step 02.** Create 4 **Class Library** projects.

## **Step 03.** Create a project (ASP.NET Core Web App (MVC).



****

Note:

* **Data Source** in this case is the SQL Server Database
* **Services Project** – This project represents a layer or component responsible for implementing the business logic of an application.
* **Repository Project** – This project provides an abstraction layer between the application’s business logic and the underlying data source.
* **Data Access Layer Project** – This project used to abstract and encapsulate the logic for accessing data from a data source, such as a database.

# Activity 02: Write codes for the BusinessObjects project

## **Step 01**. Install the following packages from NuGet:

* Microsoft.EntityFrameworkCore.SqlServer --version 8.0.2
* Microsoft.EntityFrameworkCore.Tools --version 8.0.2
* Microsoft.Extensions.Configuration.Json --version 8.0.0

Check the tool for EFCore (install/uninstall tool if needed) (dotnet SDK 8.0.202)

dotnet tool install --global dotnet-ef --version 8.0.2

dotnet tool uninstall --global dotnet-ef

## **Step 02**. Right-click on project , select **Open In Terminal.** On **Developer PowerShell** dialog execute the following commands to generate model:

* Implement ORM

dotnet ef dbcontext scaffold "Server=(local);uid=sa;pwd=12345;database=MyStore;TrustServerCertificate=True" Microsoft.EntityFrameworkCore.SqlServer --output-dir Entities

* Change the connection string in OnConfiguring() function of MyStoreContext.cs

using System.IO;

         using Microsoft.Extensions.Configuration.Json;

*private string GetConnectionString()*

*{*

*IConfiguration configuration = new ConfigurationBuilder()*

*.SetBasePath(Directory.GetCurrentDirectory())*

*.AddJsonFile("appsettings.json", true, true).Build();*

*return configuration["ConnectionStrings:DefaultConnectionString"];*

*}*

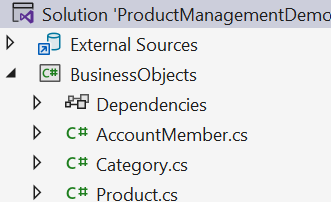
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

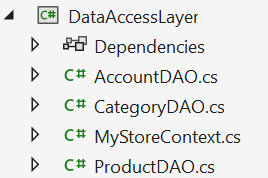
***optionsBuilder.UseSqlServer(GetConnectionString());***

         }

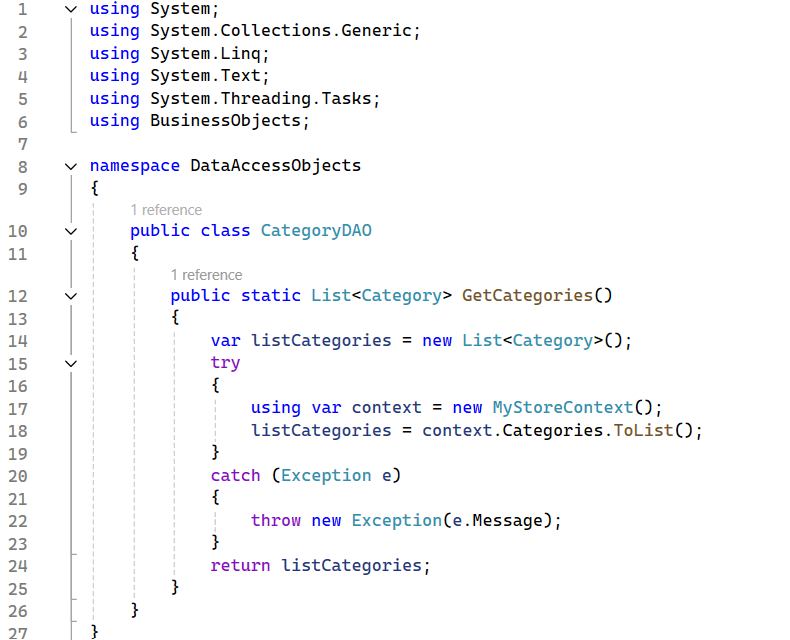
* Move the MyStoreContext.cs to DataAccessLayer Project



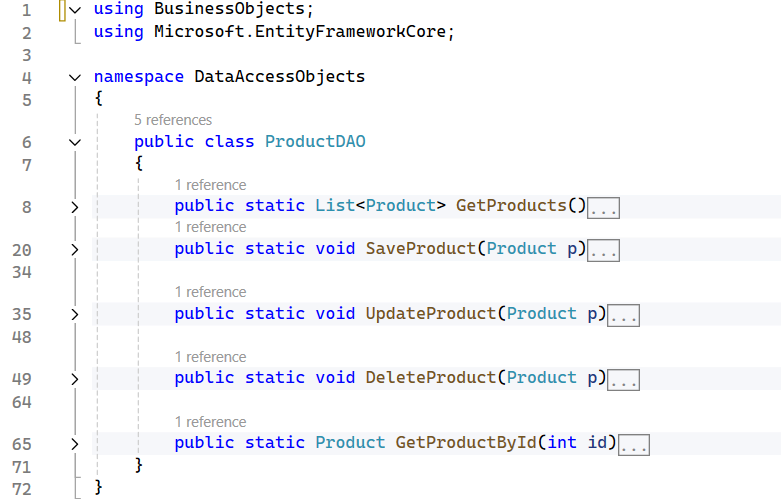
# Activity 03: Write codes for the DataAccessLayer project



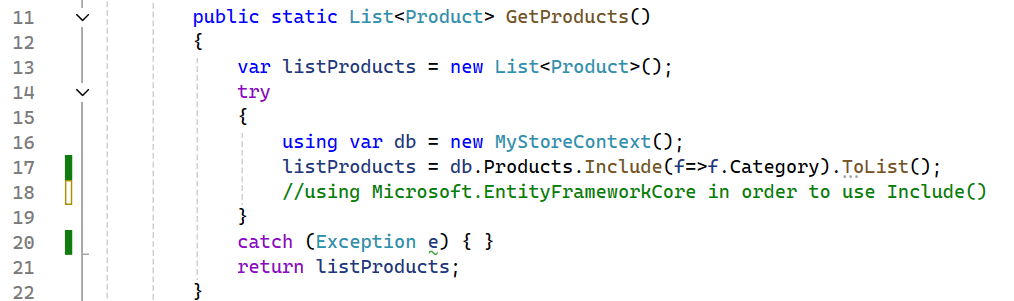
## **Step 01.** On the **DataAccessObjects** project, add a class named **CategoryDAO.cs** and write codes as follows:

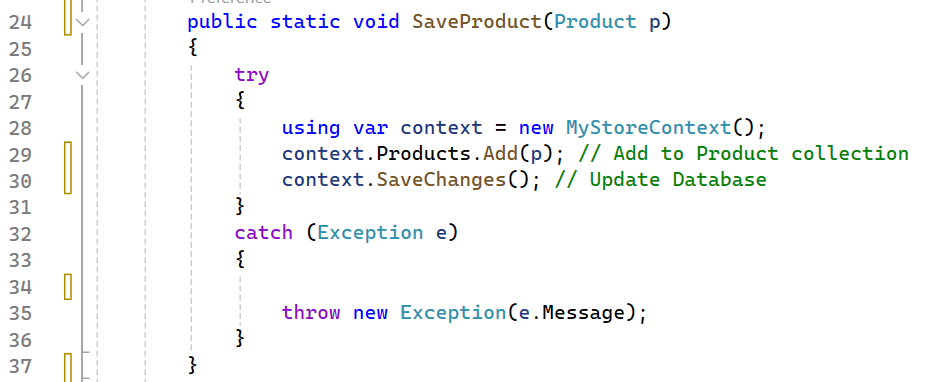


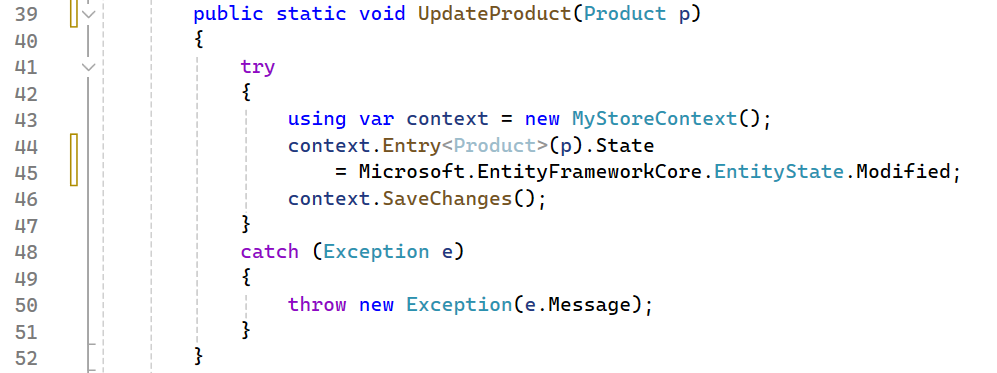
## **Step 02**. On the **DataAccessObjects** project, add a class named **ProductDAO.cs** and write codes as follows:

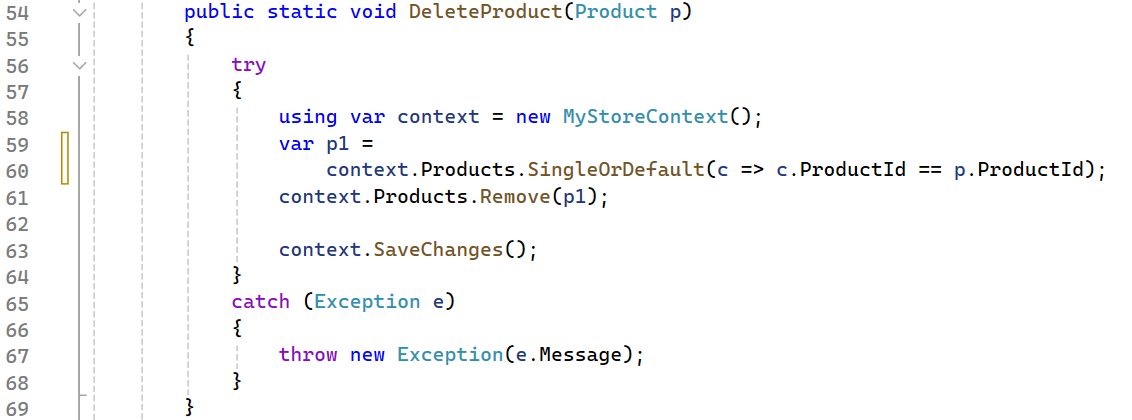


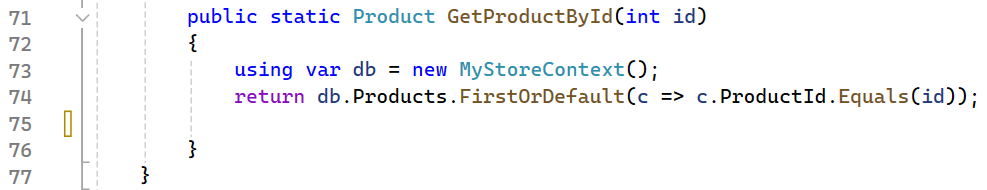
The details of functions in ProductDAO.cs:



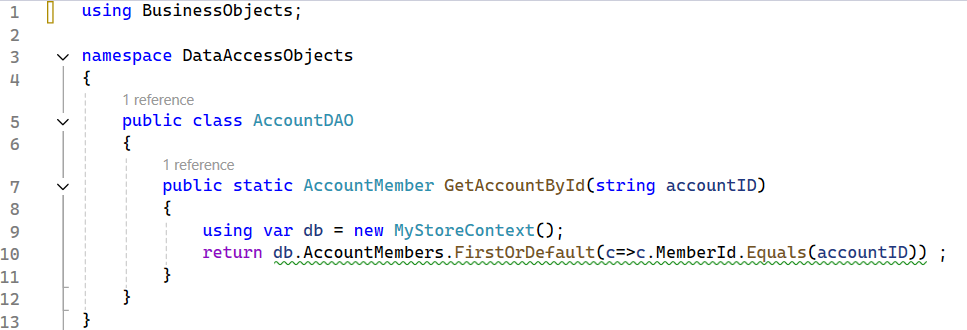




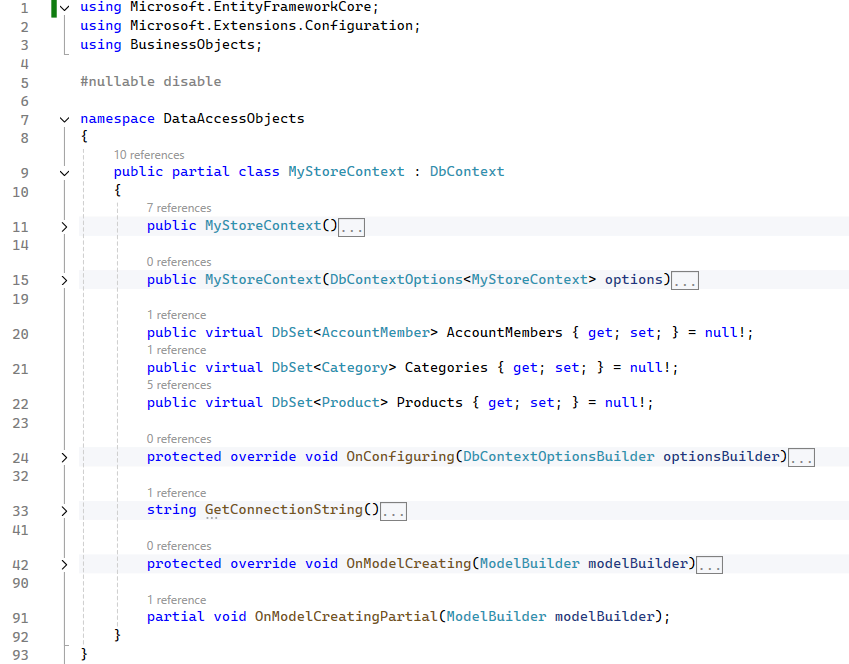




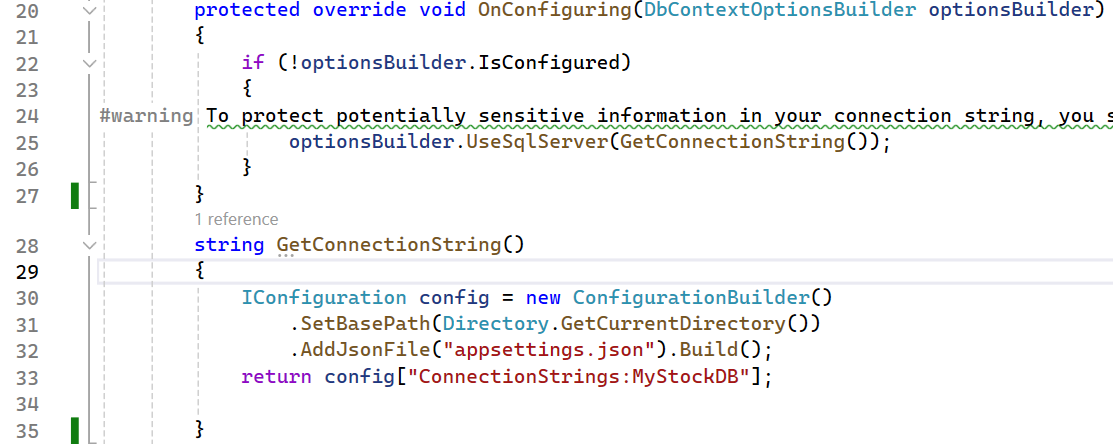
## **Step 03**. Write codes for **AccountDAO.cs** as follows:



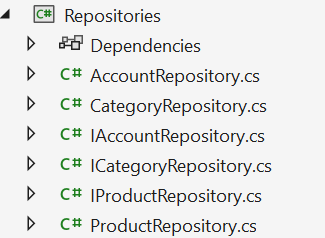
**Step 04**. The codes for **MyStoreContext.cs**:



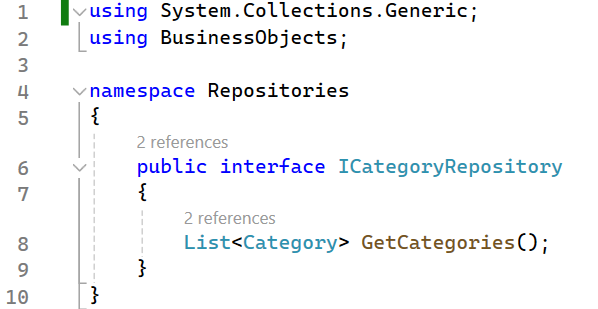
The details for GetConnectionString() and OnConfiguring() functions



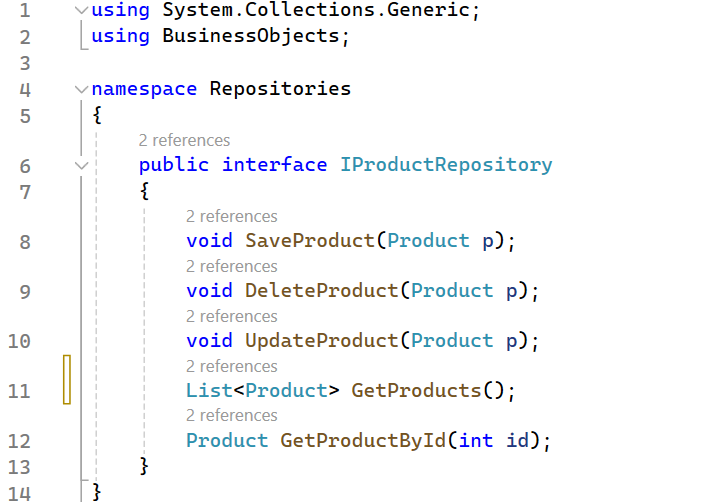
# Activity 04: Write codes for the Repositories project



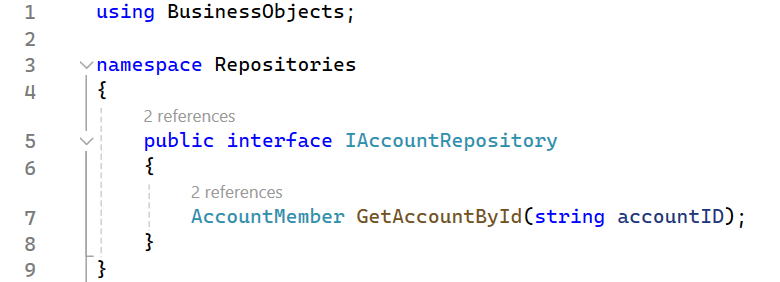
## **Step 01.** On the **Repositories** project, add an interface named **ICatergoryRepository.cs** and write codes as follows:



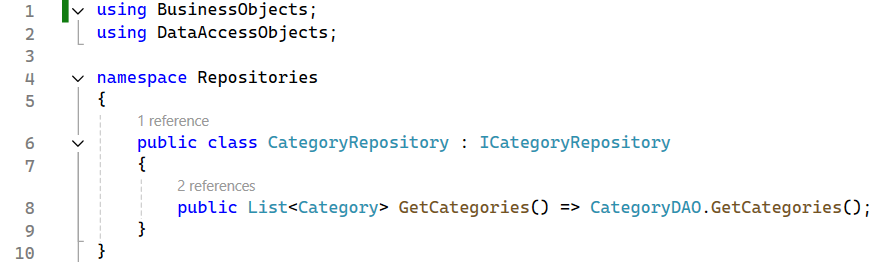
## **Step 02**. On the **Repositories** project, add an interface named **IProductRepository.cs** and write codes as follows:



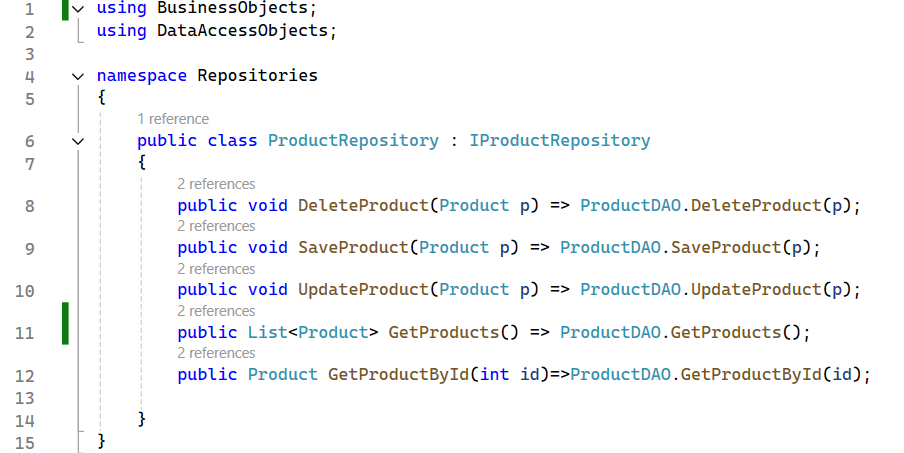
## **Step 03**. On the **Repositories** project, add an interface named **IAccountRepository.cs** and write codes as follows:



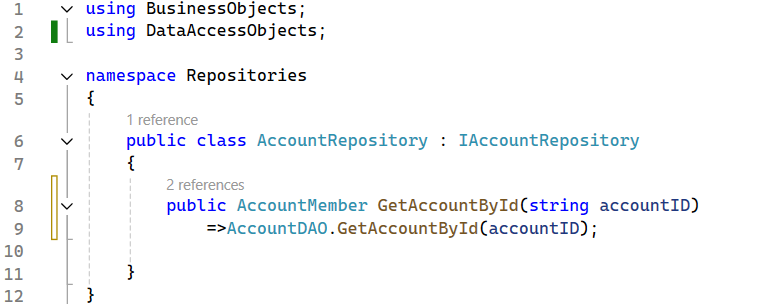
**Step 04**. Write codes for class **CategoryRepository.cs** as follows:



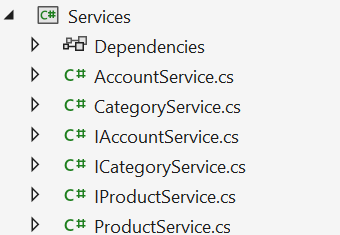
**Step 05**. Write codes for class **ProductRepository.cs** as follows:



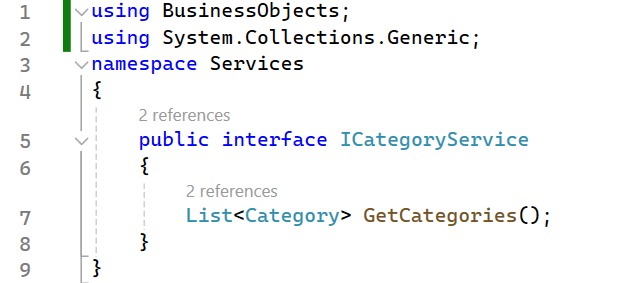
**Step 06**. Write codes for class **AccountRepository.cs** as follows:



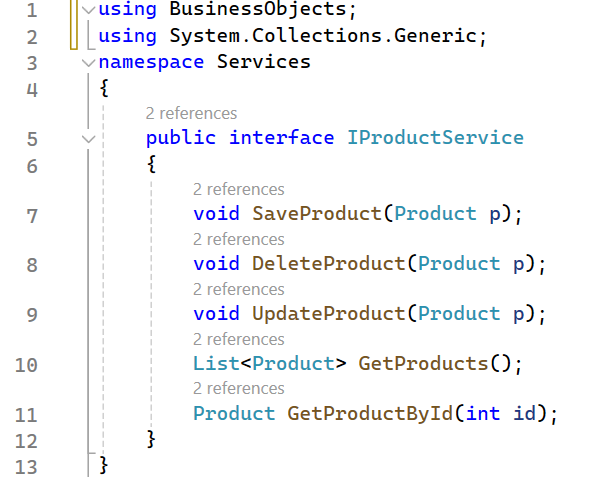
# Activity 05: Write codes for the Services project



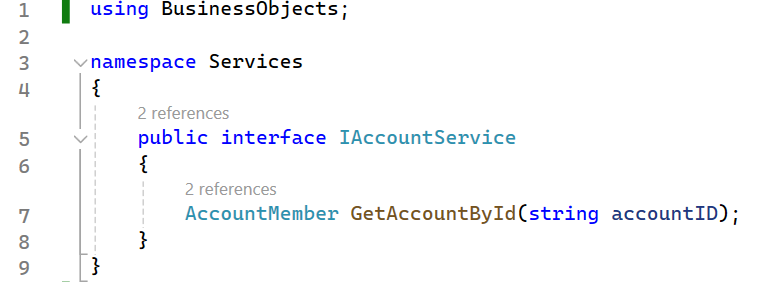
## **Step 01.** On the **Services** project, add an interface named **ICatergoryService.cs** and write codes as follows:



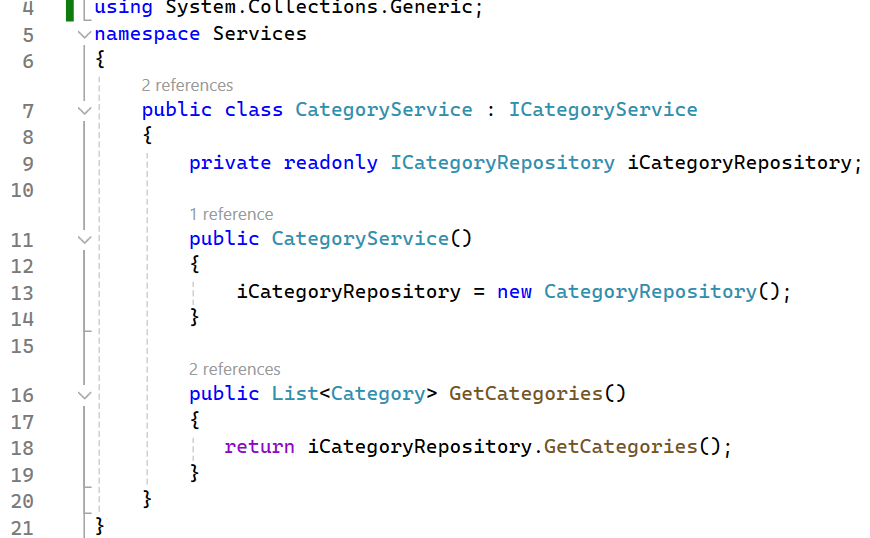
## **Step 02**. On the **Services** project, add an interface named **IProductService.cs** and write codes as follows:



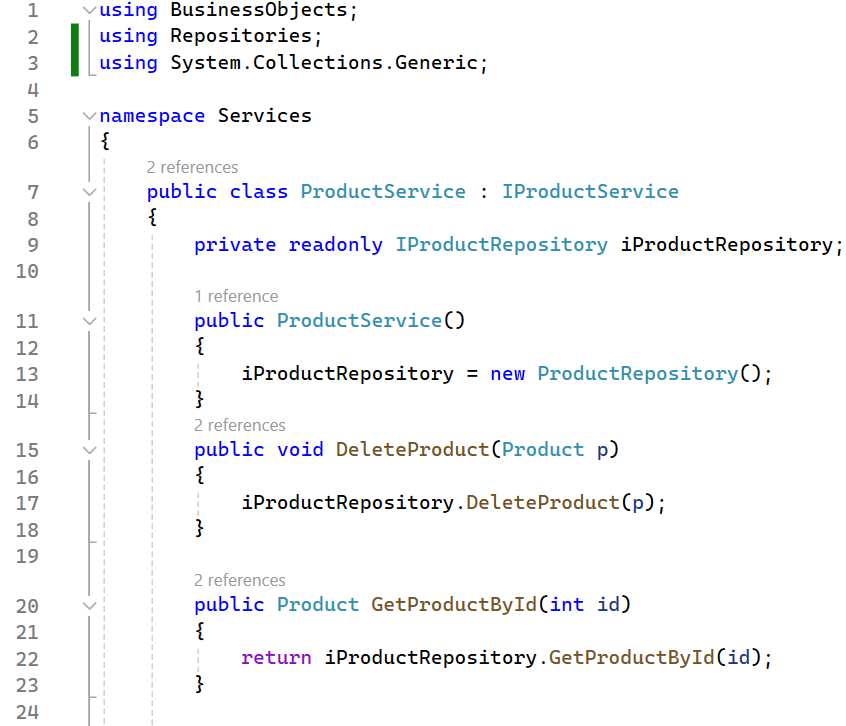
## **Step 03**. On the **Services** project, add an interface named **IAccountService.cs** and write codes as follows:

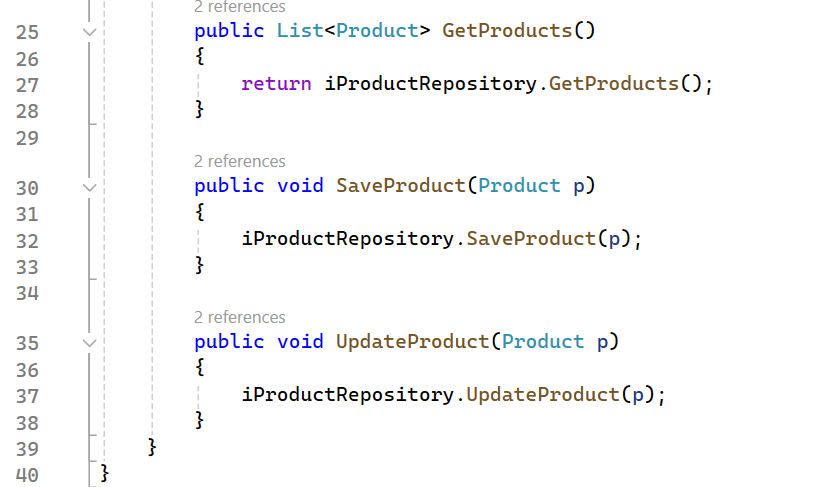


**Step 04**. Write codes for class **CategoryService.cs** as follows:

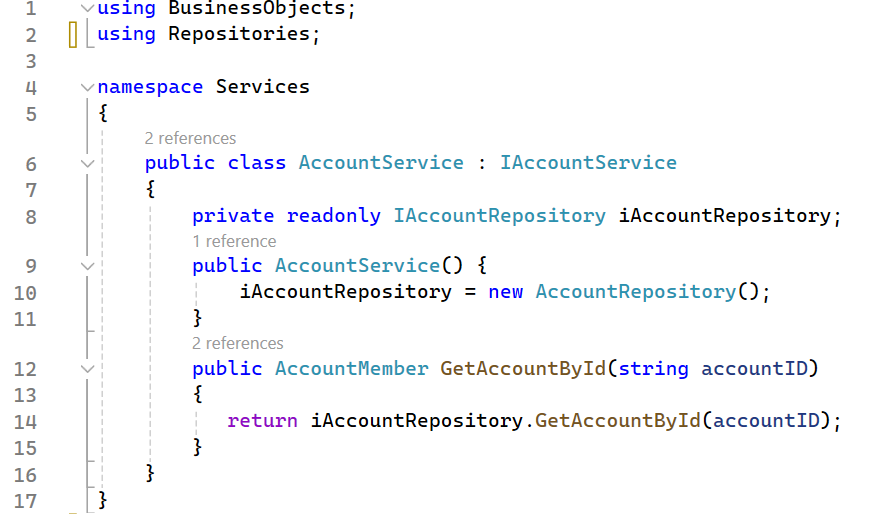


**Step 05**. Write codes for class **ProductService.cs** as follows:



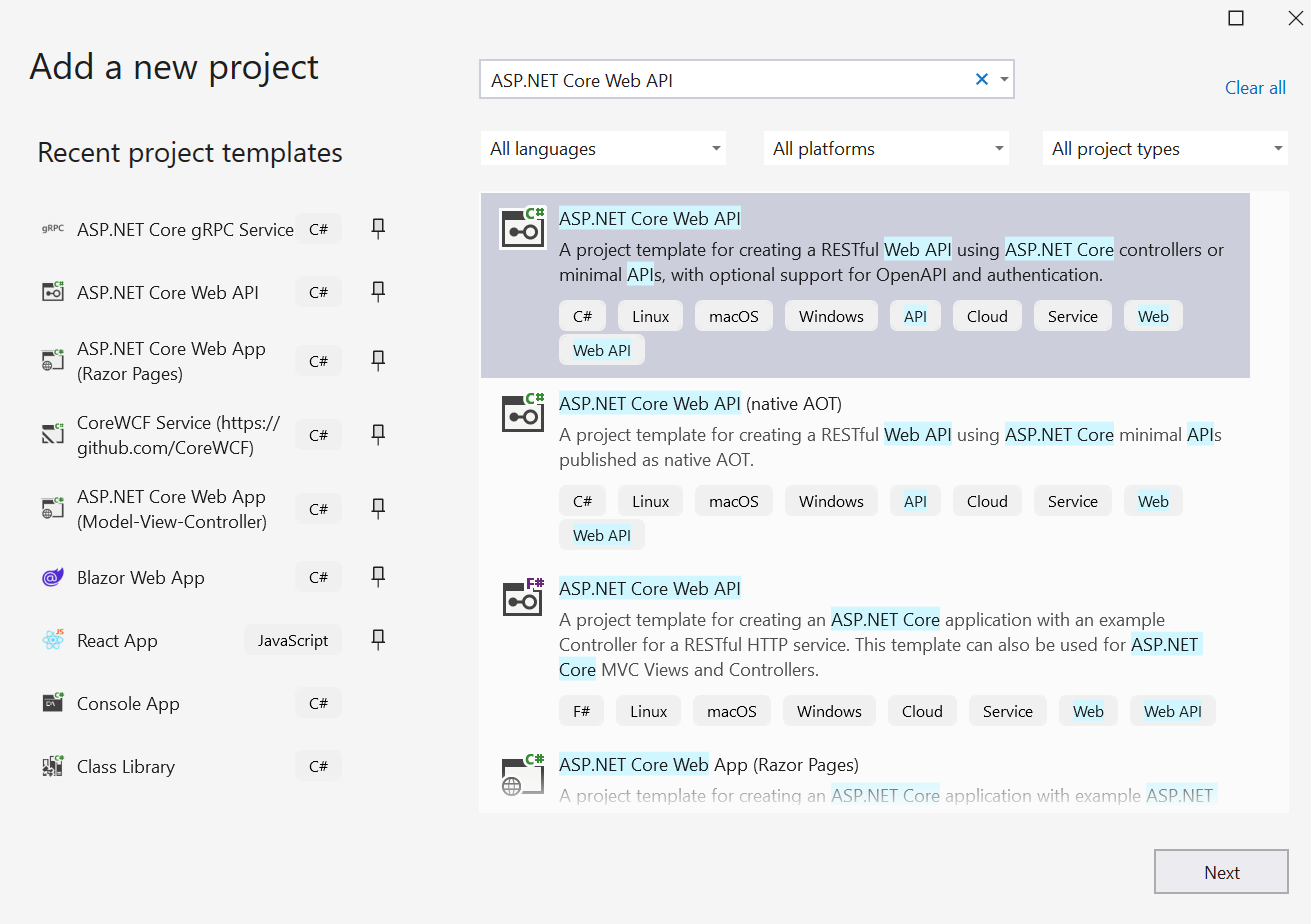


**Step 06**. Write codes for class **AccountService.cs** as follows:



# Activity 06: Work with ASP.NET Core Web API

## **Step 01.** Create the **ASP.NET Core Web API** project inside the Lab Solution

****

## **Step 02**. Add connection string to **appsettings.json**.

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

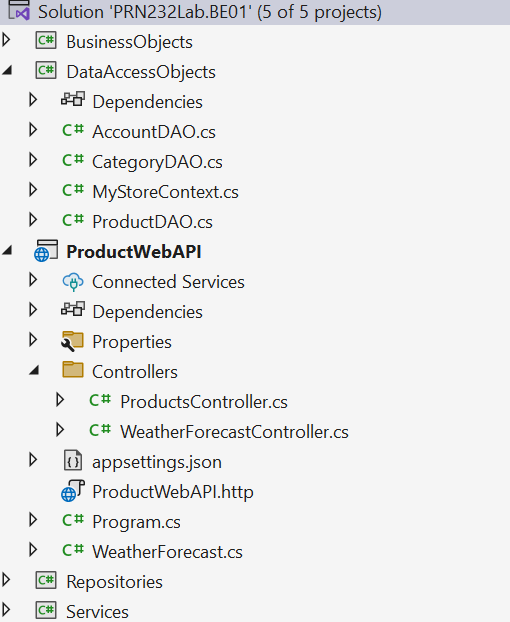
"ConnectionStrings": {

"MyStockDB": "Server=localhost;uid=sa;pwd=1234567890;database=MyStore;TrustServerCertificate=True"

}

}

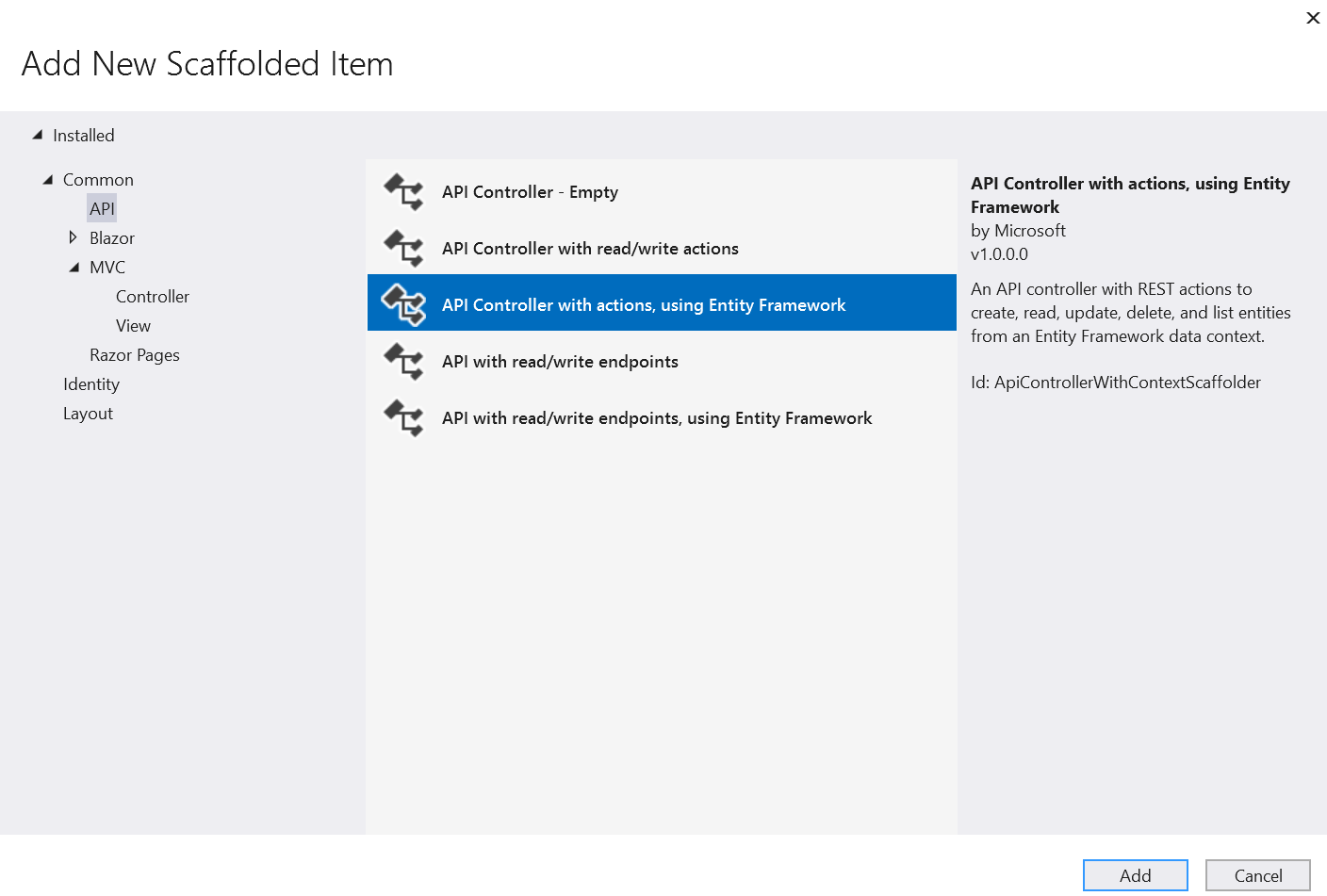
**Step 03.** Add **Business Objects** and **Services** projects as references for the ASP.NET Core Web API

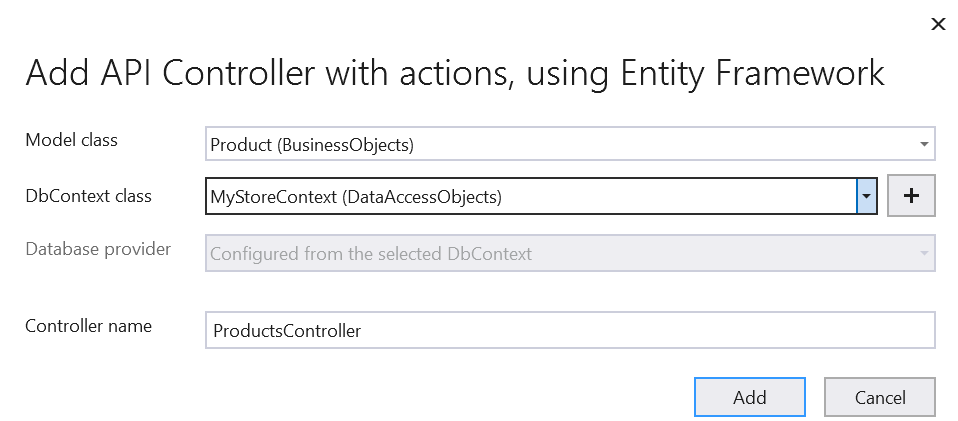
****

**Step 04.** Create Controller

* Connect direct to the Data Access Layer (MyStoreContext.cs) to generate code
* Then
  + Add Dependency Injection
  + Change the code connects to the Service Layer

Add a Controller named ProductsController.cs





## **Step 05.** After generate the ProductsController.cs of ASP.NET Core Web API project, change the code as the following

***Program.cs***

using Repositories;

using Services;

using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers().AddJsonOptions(options =>

{

options.JsonSerializerOptions.ReferenceHandler = ReferenceHandler.IgnoreCycles;

options.JsonSerializerOptions.DefaultIgnoreCondition = JsonIgnoreCondition.Never;

});

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSingleton<IProductRepository, ProductRepository>();

builder.Services.AddSingleton<IProductService, ProductService>();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseAuthorization();

app.MapControllers();

app.Run();

***ProductsController.cs***

******

*Use Dependency Injection for IProductService,*

private readonly IProductService \_context;

public ProductsController(IProductService context)

{

\_context = context;

}

*Action method for get all information of products of ProductsController.cs (Method GET)*

// GET: api/Products

[HttpGet]

public async Task<ActionResult<IEnumerable<Product>>> GetProducts()

{

return \_context.GetProducts();

}

*Action method for get product by ID of ProductsController.cs (Method GET: Products/Detail/5)*

// GET: api/Products/5

[HttpGet("{id}")]

public async Task<ActionResult<Product>> GetProduct(int id)

{

var product = \_context.GetProductById(id);

if (product == null)

{

return NotFound();

}

return product;

}

*Action method for creating a new product of ProductsController.cs (Method POST: Products/Create)*

using Microsoft.AspNetCore.Mvc;

using Microsoft.EntityFrameworkCore;

using BusinessObjects;

using Services;

namespace ProductWebAPI.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ProductsController : ControllerBase

{

private readonly IProductService \_context;

public ProductsController(IProductService context)

{

\_context = context;

}

// GET: api/Products

[HttpGet]

public async Task<ActionResult<IEnumerable<Product>>> GetProducts()

{

return \_context.GetProducts();

}

// GET: api/Products/5

[HttpGet("{id}")]

public async Task<ActionResult<Product>> GetProduct(int id)

{

var product = \_context.GetProductById(id);

if (product == null)

{

return NotFound();

}

return product;

}

// PUT: api/Products/5

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPut("{id}")]

public async Task<IActionResult> PutProduct(int id, Product product)

{

if (id != product.ProductId)

{

return BadRequest();

}

try

{

\_context.UpdateProduct(product);

}

catch (DbUpdateConcurrencyException)

{

if (\_context.GetProductById(id) == null)

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

// POST: api/Products

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPost]

public async Task<ActionResult<Product>> PostProduct(Product product)

{

\_context.SaveProduct(product);

return CreatedAtAction("GetProduct", new { id = product.ProductId }, product);

}

// DELETE: api/Products/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteProduct(int id)

{

var product = \_context.GetProductById(id);

if (product == null)

{

return NotFound();

}

\_context.DeleteProduct(product);

return NoContent();

}

}

}

*Action method for opening a form to edit an existing product of ProductsController.cs (Method PUT: Products/Edit/5)*

// PUT: api/Products/5

// To protect from overposting attacks, see https://go.microsoft.com/fwlink/?linkid=2123754

[HttpPut("{id}")]

public async Task<IActionResult> PutProduct(int id, Product product)

{

if (id != product.ProductId)

{

return BadRequest();

}

try

{

\_context.UpdateProduct(product);

}

catch (DbUpdateConcurrencyException)

{

if (\_context.GetProductById(id) == null)

{

return NotFound();

}

else

{

throw;

}

}

return NoContent();

}

*Action method for deleting an existing product of ProductsController.cs (Method POST: Products/Delete/5)*

// DELETE: api/Products/5

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteProduct(int id)

{

var product = \_context.GetProductById(id);

if (product == null)

{

return NotFound();

}

\_context.DeleteProduct(product);

return NoContent();

}

## **Step 06.** Clean ASP.NET Core Web API project, remove Entity Framework Core related packages.

# Activity 07: Run the ASP.NET Core Web App (MVC) project and test all actions