Building Product Management Application Using ASP.NET Core Web API

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, CategoryId, 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 Application with Model-View-Controller. A **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.
- Use Entity Framework 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 **Lab01_ASP.NETCoreWebAPI**.

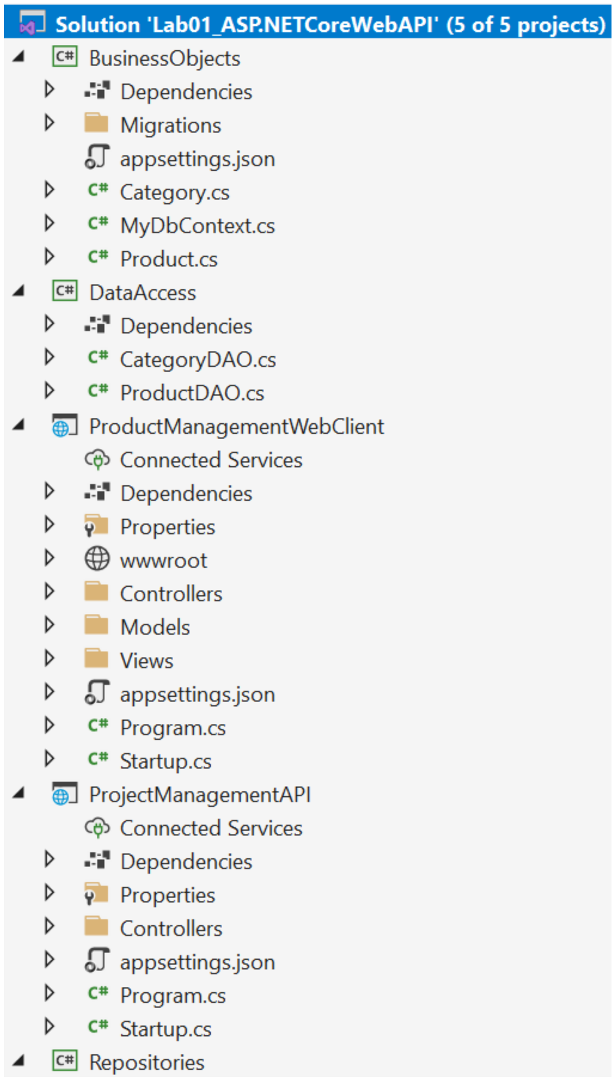
Step 02. Create Class Library Project: BusinessObjects.

Step 03. Create Class Library Project: Repositories.

Step 04. Create Class Library Project: DataAccess.

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

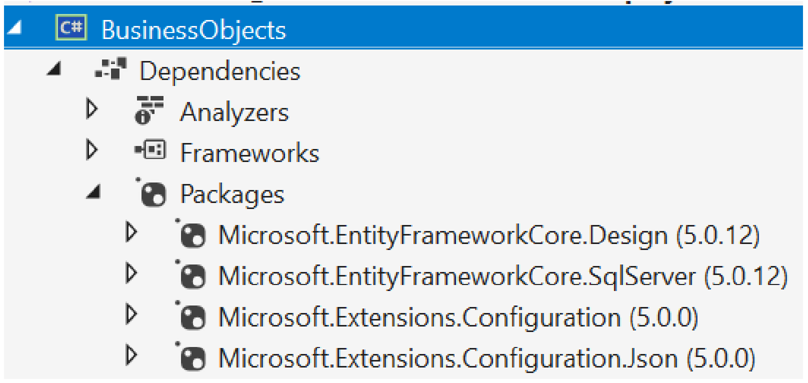
Step 06. Create ASP.NET Core Web Application (Model-View-Controller) Project.



Activity 02: BusinessObjects Project - Work with Entity Framework

Step 01. Create Class Library Project named BusinessObjects

Step 02. Install the following packages from NuGet:



```
◢ BusinessObjects
   ◢ Dependencies
      ▷ Analyzers
      ▷ Frameworks
      ◢ Packages
         ▷ Microsoft.EntityFrameworkCore.Design (5.0.12)
         ▷ Microsoft.EntityFrameworkCore.SqlServer (5.0.12)
         ▷ Microsoft.Extensions.Configuration (5.0.0)
         ▷ Microsoft.Extensions.Configuration.Json (5.0.0)
```

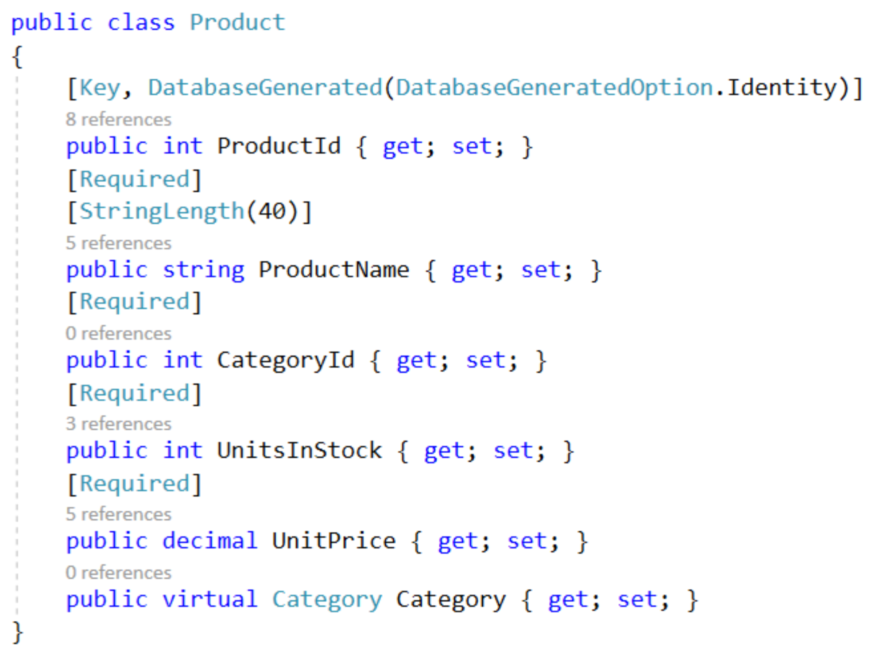
Step 03. Add Connection string (also add JSON appsettings.json file)

```json
{
  "ConnectionStrings": {
    "MyStoreDB": "Server=(local);Uid=sa;Pwd=1234567890;Database=MyStoreDB"
  }
}
```

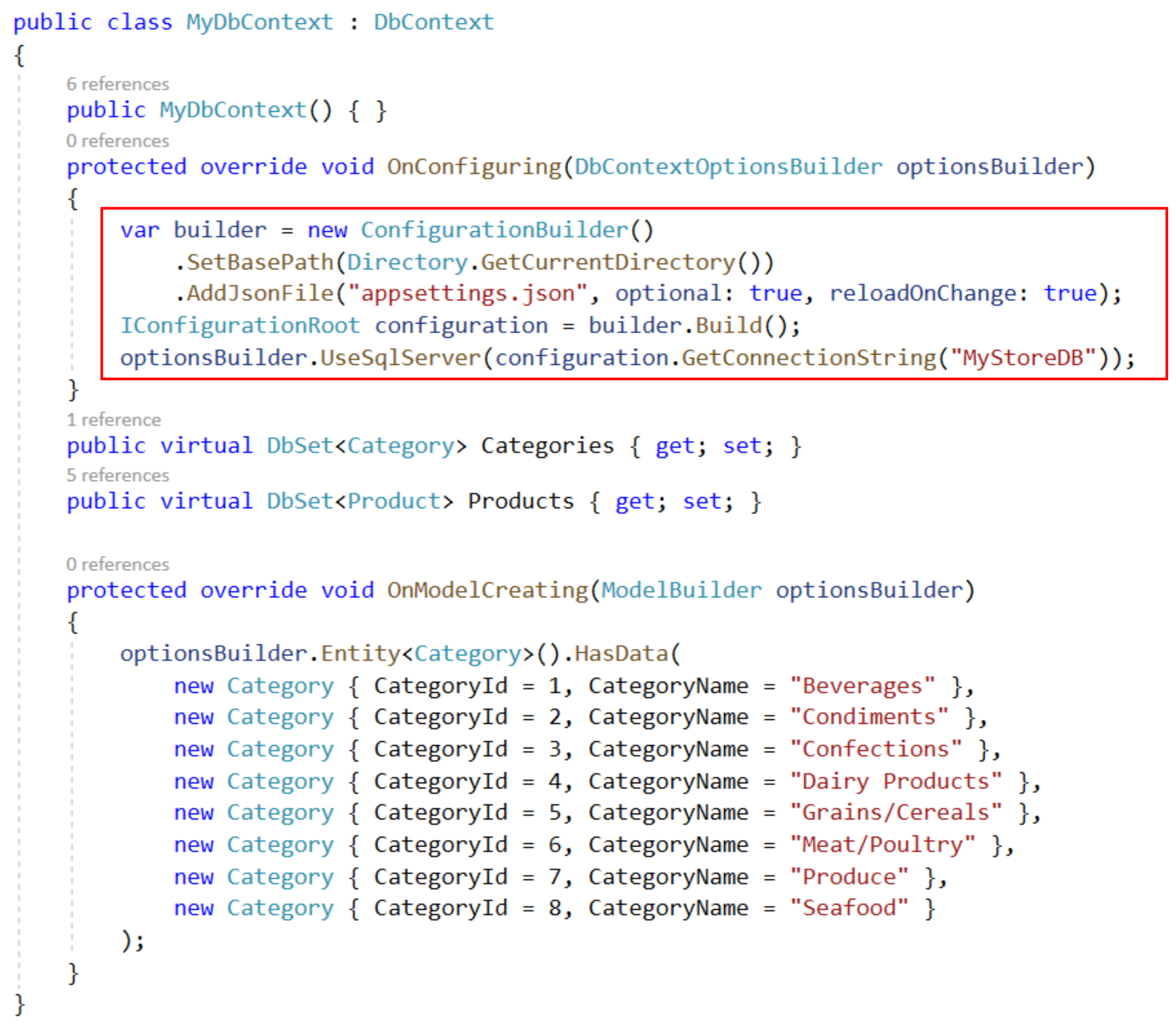
```xml
<ItemGroup>
  <None Update="appsettings.json">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </None>
</ItemGroup>
```

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

```csharp
public class Category
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    8 references
    public int CategoryId { get; set; }
    [Required]
    [StringLength(40)]
    8 references
    public string CategoryName { get; set; }
    0 references
    public virtual ICollection<Product> Products { get; set; }
}
```



```
public class Product
{
    [Key, DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    8 references
    public int ProductId { get; set; }
    [Required]
    [StringLength(40)]
    5 references
    public string ProductName { get; set; }
    [Required]
    0 references
    public int CategoryId { get; set; }
    [Required]
    3 references
    public int UnitsInStock { get; set; }
    [Required]
    5 references
    public decimal UnitPrice { get; set; }
    0 references
    public virtual Category Category { get; set; }
}
```



```
public class MyDbContext : DbContext
{
    6 references
    public MyDbContext() { }
    0 references
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        var builder = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("appsettings.json", optional: true, reloadOnChange: true);
        IConfigurationRoot configuration = builder.Build();
        optionsBuilder.UseSqlServer(configuration.GetConnectionString("MyStoreDB"));
    }
    1 reference
    public virtual DbSet<Category> Categories { get; set; }
    5 references
    public virtual DbSet<Product> Products { get; set; }

    0 references
    protected override void OnModelCreating(ModelBuilder optionsBuilder)
    {
        optionsBuilder.Entity<Category>().HasData(
            new Category { CategoryId = 1, CategoryName = "Beverages" },
            new Category { CategoryId = 2, CategoryName = "Condiments" },
            new Category { CategoryId = 3, CategoryName = "Confections" },
            new Category { CategoryId = 4, CategoryName = "Dairy Products" },
            new Category { CategoryId = 5, CategoryName = "Grains/Cereals" },
            new Category { CategoryId = 6, CategoryName = "Meat/Poultry" },
            new Category { CategoryId = 7, CategoryName = "Produce" },
            new Category { CategoryId = 8, CategoryName = "Seafood" }
        );
    }
}
```

Step 05. Add-Migration and Update-Database

dotnet ef migrations add "InitialDB"

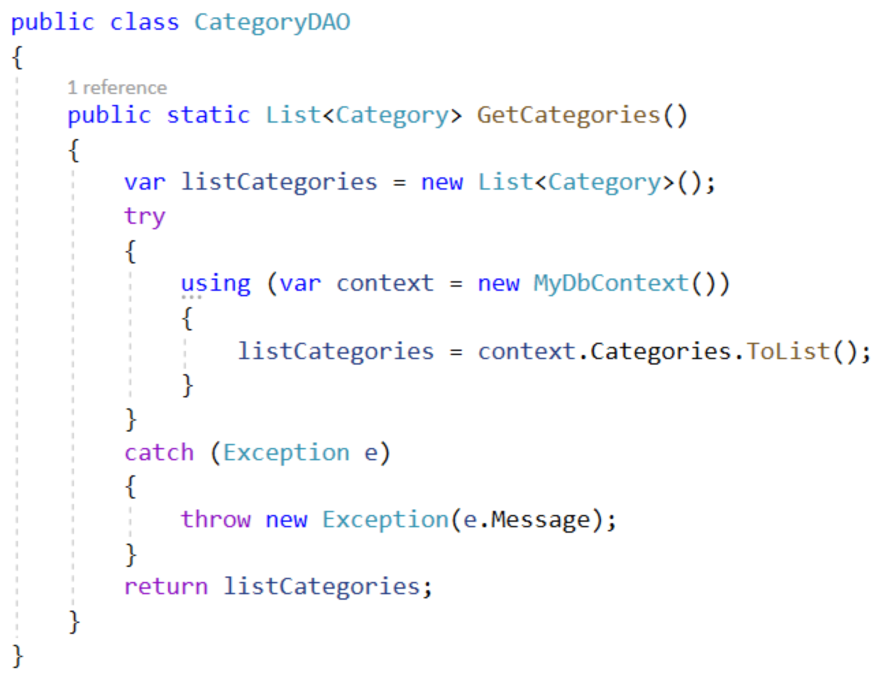
dotnet ef database update

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

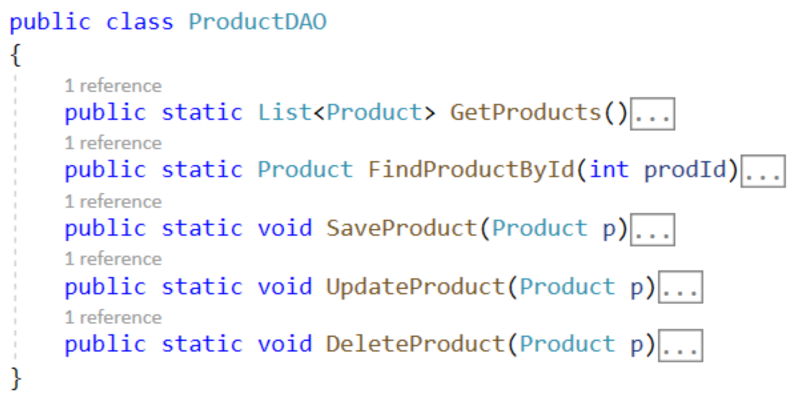
Step 01. Create Class Library Project named DataAccess

Step 02. Add Project reference: BusinessObjects Project

Step 03. Add data access classes for Product and Category



```
public class CategoryDAO
{
    1 reference
    public static List<Category> GetCategories()
    {
        var listCategories = new List<Category>();
        try
        {
            using (var context = new MyDbContext())
            {
                listCategories = context.Categories.ToList();
            }
        }
        catch (Exception e)
        {
            throw new Exception(e.Message);
        }
        return listCategories;
    }
}
```



```
public class ProductDAO
{
    1 reference
    public static List<Product> GetProducts()...
    1 reference
    public static Product FindProductById(int prodId)...
    1 reference
    public static void SaveProduct(Product p)...
    1 reference
    public static void UpdateProduct(Product p)...
    1 reference
    public static void DeleteProduct(Product p)...
}
```

The detail of functions ProductDAO.cs

```
public static List<Product> GetProducts()
{
    var listProducts = new List<Product>();
    try
    {
        using (var context = new MyDbContext())
        {
            listProducts = context.Products.ToList();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
    return listProducts;
}

public static Product FindProductById(int prodId)
{
    Product p = new Product();
    try
    {
        using (var context = new MyDbContext())
        {
            p = context.Products.SingleOrDefault(x=> x.ProductId==prodId);
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
    return p;
}

public static void SaveProduct(Product p)
{
    try
    {
        using (var context = new MyDbContext())
        {
            context.Products.Add(p);
            context.SaveChanges();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}
```

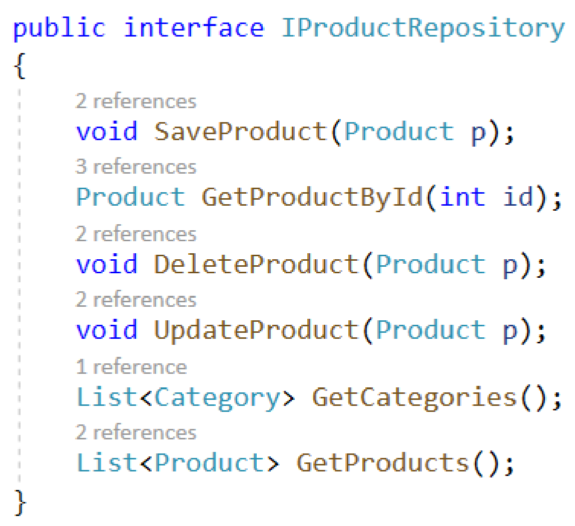
```
public static void UpdateProduct(Product p)
{
    try
    {
        using (var context = new MyDbContext())
        {
            context.Entry<Product>(p).State =
                Microsoft.EntityFrameworkCore.EntityState.Modified;
            context.SaveChanges();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}

public static void DeleteProduct(Product p)
{
    try
    {
        using (var context = new MyDbContext())
        {
            var p1 = context.Products.SingleOrDefault(
                                c => c.ProductId == p.ProductId);
            context.Products.Remove(p1);

            context.SaveChanges();
        }
    }
    catch (Exception e)
    {
        throw new Exception(e.Message);
    }
}
```

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

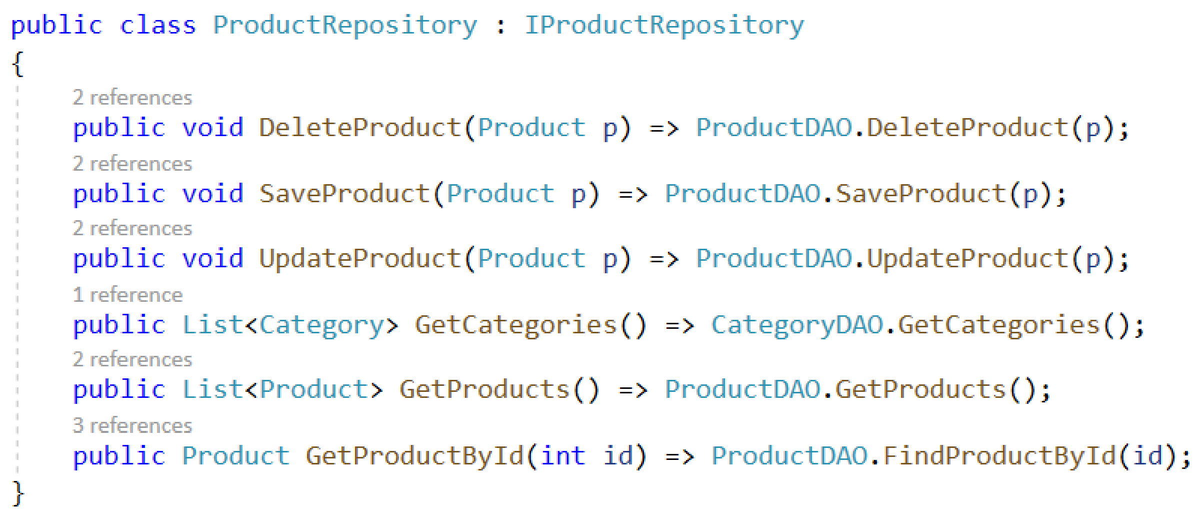
Step 02. Add Project reference: BusinessObjects, DataAccess Projects

Step 03. Create IProductRepository Interface



```
public interface IProductRepository
{
    2 references
    void SaveProduct(Product p);
    3 references
    Product GetProductById(int id);
    2 references
    void DeleteProduct(Product p);
    2 references
    void UpdateProduct(Product p);
    1 reference
    List<Category> GetCategories();
    2 references
    List<Product> GetProducts();
}
```

Step 04. Create ProductRepository class implements IProductRepository Interface



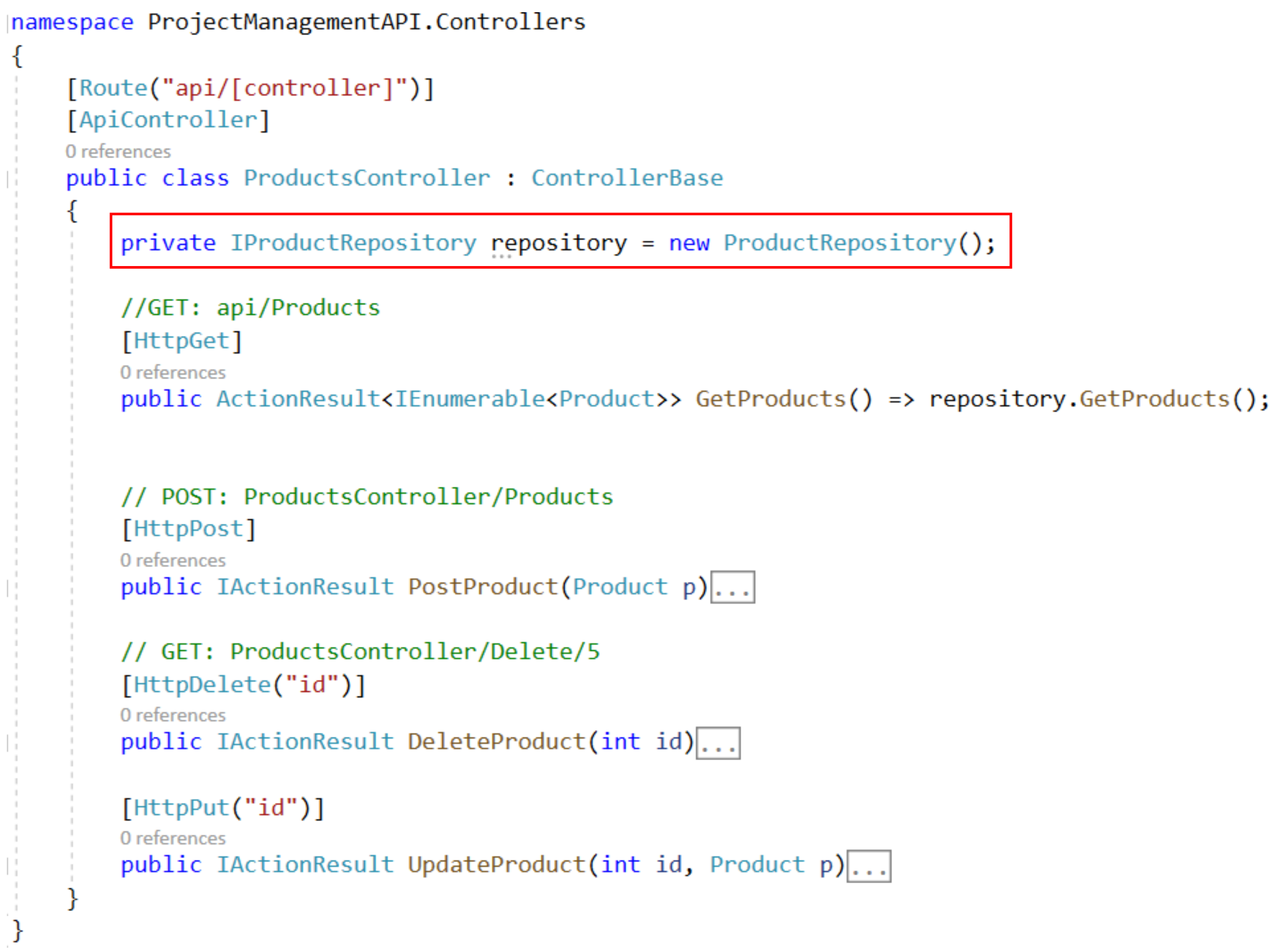
```
public class ProductRepository : IProductRepository
{
    2 references
    public void DeleteProduct(Product p) => ProductDAO.DeleteProduct(p);
    2 references
    public void SaveProduct(Product p) => ProductDAO.SaveProduct(p);
    2 references
    public void UpdateProduct(Product p) => ProductDAO.UpdateProduct(p);
    1 reference
    public List<Category> GetCategories() => CategoryDAO.GetCategories();
    2 references
    public List<Product> GetProducts() => ProductDAO.GetProducts();
    3 references
    public Product GetProductById(int id) => ProductDAO.FindProductById(id);
}
```

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

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

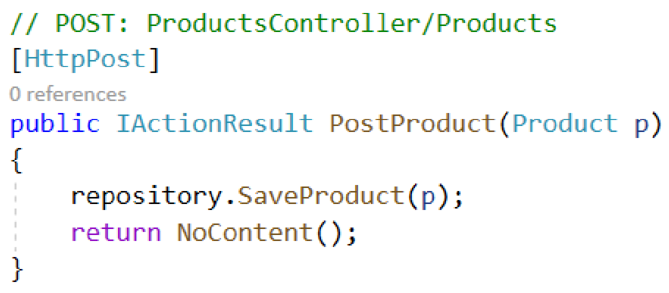
Step 02. Add Project reference: Repository Project

Step 03. Add ApiController named ProductsControllers.cs



```
namespace ProjectManagementAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    0 references
    public class ProductsController : ControllerBase
    {
        private IProductRepository repository = new ProductRepository();

        //GET: api/Products
        [HttpGet]
        0 references
        public ActionResult<IEnumerable<Product>> GetProducts() => repository.GetProducts();


        // POST: ProductsController/Products
        [HttpPost]
        0 references
        public IActionResult PostProduct(Product p)...

        // GET: ProductsController/Delete/5
        [HttpDelete("id")]
        0 references
        public IActionResult DeleteProduct(int id)...

        [HttpPut("id")]
        0 references
        public IActionResult UpdateProduct(int id, Product p)...
    }
}
```

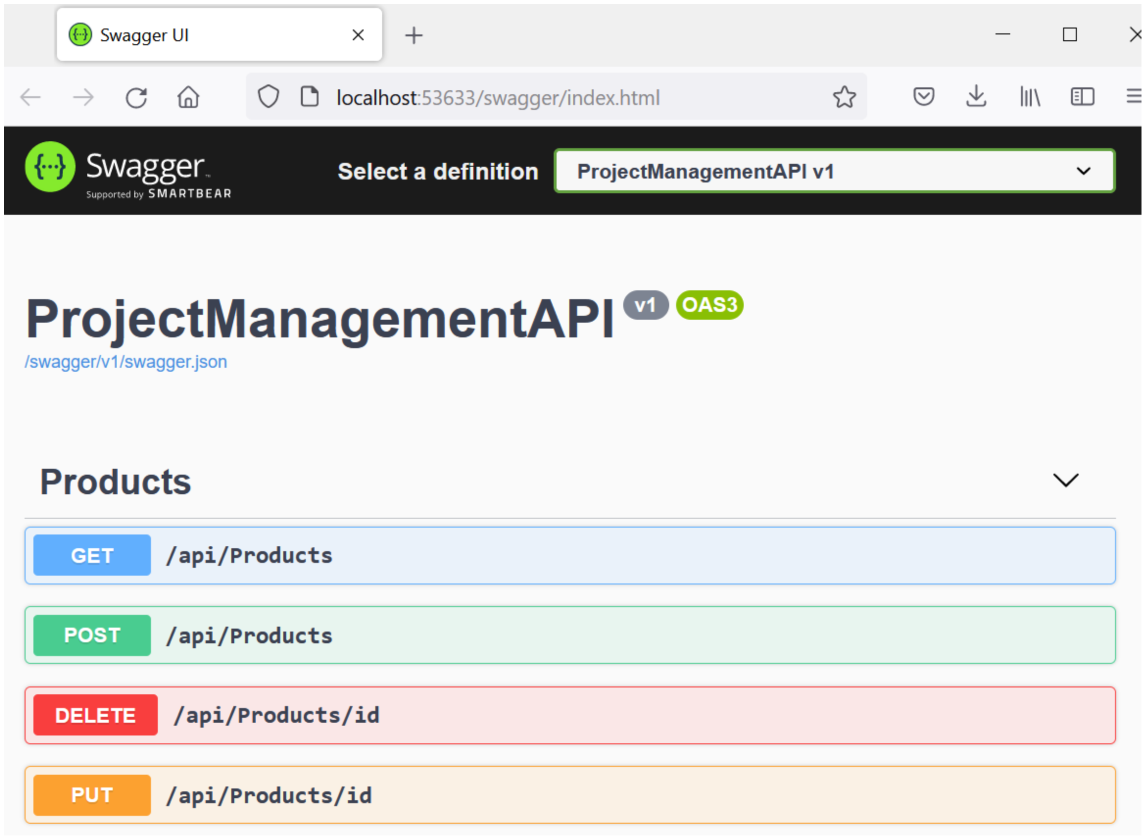
The detail of functions in ProductControllers (Web API).



```
// POST: ProductsController/Products
[HttpPost]
0 references
public IActionResult PostProduct(Product p)
{
    repository.SaveProduct(p);
    return NoContent();
}
```

```
// GET: ProductsController/Delete/5
[HttpDelete("id")]
0 references
public IActionResult DeleteProduct(int id)
{
    var p = repository.GetProductById(id);
    if (p == null)
        return NotFound();
    repository.DeleteProduct(p);
    return NoContent();
}

[HttpPut("id")]
0 references
public IActionResult UpdateProduct(int id, Product p)
{
    var pTmp = repository.GetProductById(id);
    if (p == null)
        return NotFound();
    repository.UpdateProduct(p);
    return NoContent();
}
```

<u>**Step 04**</u>. Test API project with OpenAPI or Postman



ProjectManagementAPI v1 OAS3
/swagger/v1/swagger.json

Products

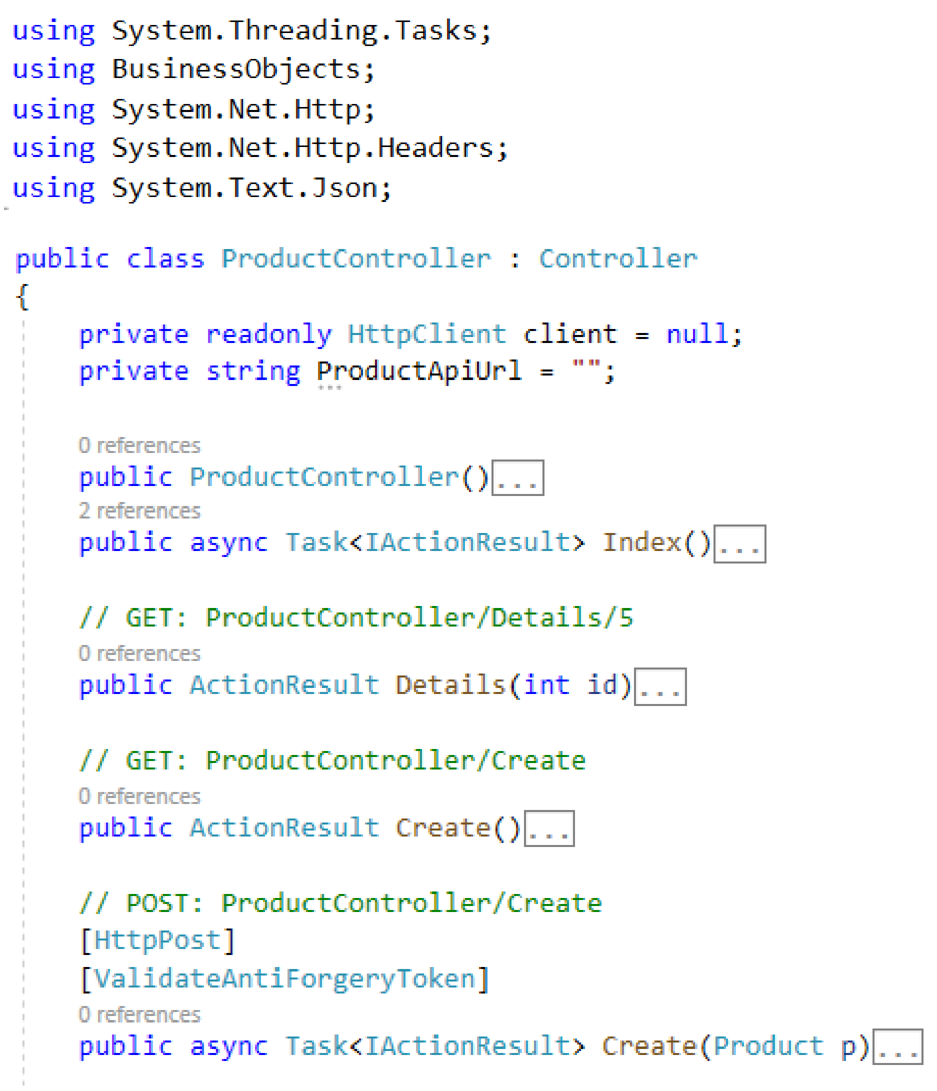
GET /api/Products
POST /api/Products
DELETE /api/Products/id
PUT /api/Products/id

Activity 06: ASP.NET Core Web Application with Model-View-Controller Project

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

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

Step 03. Create Controller to connect to ProductManagementAPI



```
using System.Threading.Tasks;
using BusinessObjects;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text.Json;

public class ProductController : Controller
{
    private readonly HttpClient client = null;
    private string ProductApiUrl = "";

    0 references
    public ProductController()...
    2 references
    public async Task<IActionResult> Index()...

    // GET: ProductController/Details/5
    0 references
    public ActionResult Details(int id)...

    // GET: ProductController/Create
    0 references
    public ActionResult Create()...

    // POST: ProductController/Create
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public async Task<IActionResult> Create(Product p)...
    // GET: ProductController/Edit/5
    0 references
    public ActionResult Edit(int id)...

    // POST: ProductController/Edit/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Edit(int id, IFormCollection collection)...

    // GET: ProductController/Delete/5
    0 references
    public ActionResult Delete(int id)...

    // POST: ProductController/Delete/5
    [HttpPost]
    [ValidateAntiForgeryToken]
    0 references
    public ActionResult Delete(int id, IFormCollection collection)...
}
```

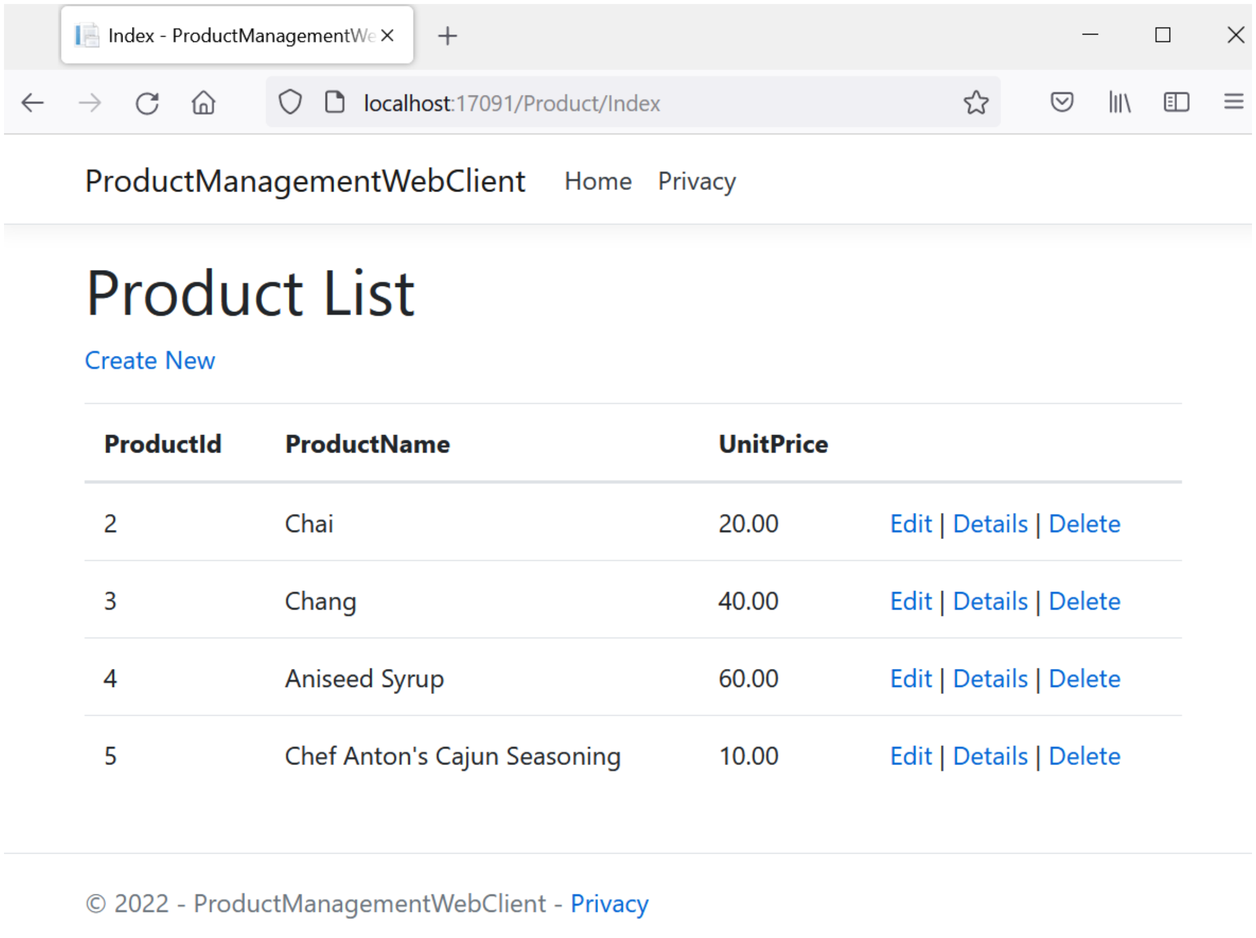
The detail of functions in ProductController (Web App MVC).

```
public ProductController()
{
    client = new HttpClient();
    var contentType = new MediaTypeWithQualityHeaderValue("application/json");
    client.DefaultRequestHeaders.Accept.Add(contentType);
    ProductApiUrl = "http://localhost:53633/api/products";
}

public async Task<IActionResult> Index()
{
    HttpResponseMessage response = await client.GetAsync(ProductApiUrl);
    string strData = await response.Content.ReadAsStringAsync();

    var options = new JsonSerializerOptions
    {
        PropertyNameCaseInsensitive = true
    };
    List<Product> listProducts = JsonSerializer.Deserialize<List<Product>>(strData, options);
    return View(listProducts);
}
```

Step 04. Create View

```
@model IEnumerable<BusinessObjects.Product>

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.ProductId)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.ProductName)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.UnitPrice)
            </th>
            <th></th>
        </tr>
    </thead>

   <tbody>
      @foreach (var item in Model)
      {
          <tr>
              <td>
                  @Html.DisplayFor(modelItem => item.ProductId)
              </td>
              <td>
                  @Html.DisplayFor(modelItem => item.ProductName)
              </td>
              <td>
                  @Html.DisplayFor(modelItem => item.UnitPrice)
              </td>
              <td>
                  @Html.ActionLink("Details", "Details", new { /* id=item.PrimaryKey */ }) |
                  @Html.ActionLink("Delete", "Delete", new { /* id=item.PrimaryKey */ })
              </td>
          </tr>
      }
   </tbody>
</table>
```

Step 05. Test the function of Web Client



Index - ProductManagementWe

localhost:17091/Product/Index

ProductManagementWebClient Home Privacy

Product List

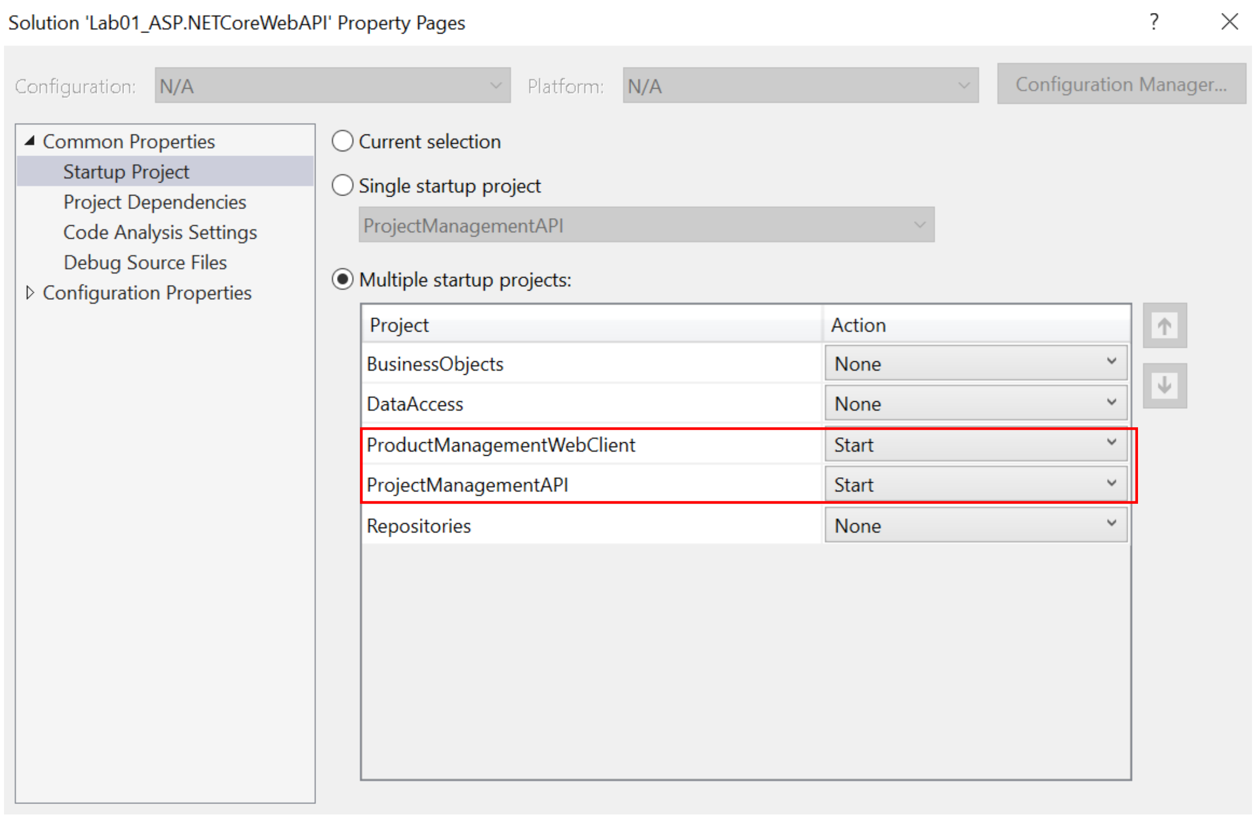
Create New

ProductId	ProductName	UnitPrice	
2	Chai	20.00	Edit | Details | Delete
3	Chang	40.00	Edit | Details | Delete
4	Aniseed Syrup	60.00	Edit | Details | Delete
5	Chef Anton's Cajun Seasoning	10.00	Edit | Details | Delete

© 2022 - ProductManagementWebClient - Privacy

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

Note: Choose the option for multiple startup projects.



Solution 'Lab01_ASP.NETCoreWebAPI' Property Pages

Configuration: N/A Platform: N/A

Current selection [✓ Multiple startup projects:] Single startup project (ProjectManagementAPI)

Project	Action
BusinessObjects	None
DataAccess	None
ProductManagementWebClient	Start
ProjectManagementAPI	Start
Repositories	None