**Lab 02. Building an Product Management Application using ASP.NET Core Web App Razor Page and SignalR**

# 1. Introduction

Imagine you're an employee of a store named **ProductStore**. Your manager has asked you to develop a Web 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 (Razor Pages). 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 App Razor Pages** 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 Razor Pages
* Work with Realtime Communication Web application
* Run the project and test the application actions.

# 3. Database Design (MyStore)

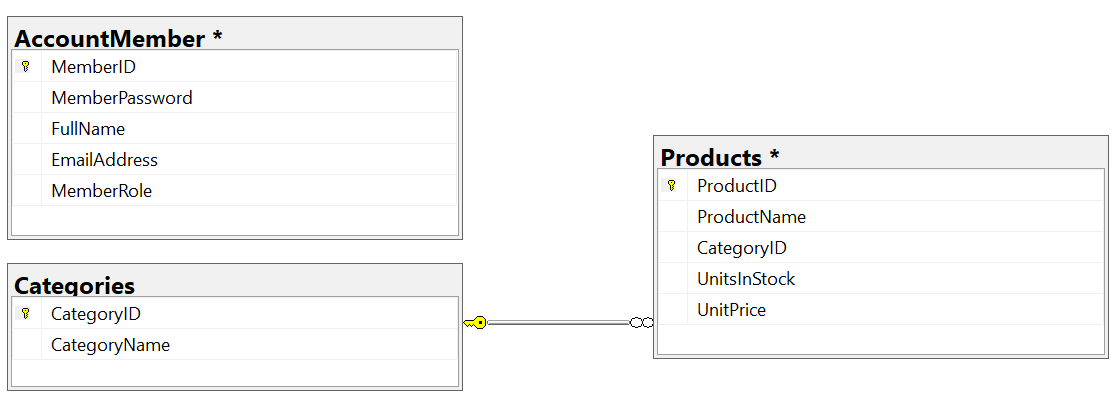


Table AccountMember

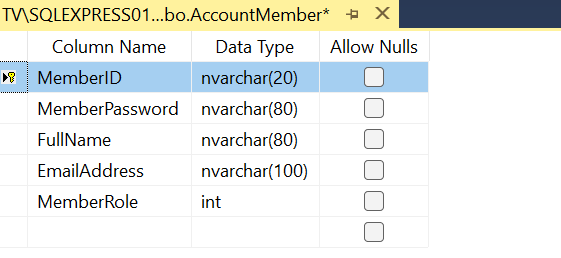


Table Categories

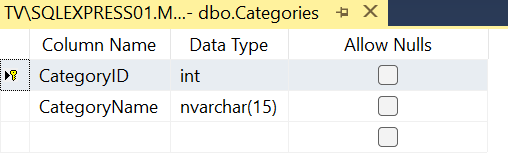
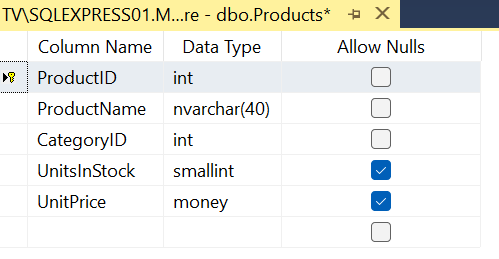


Table Products



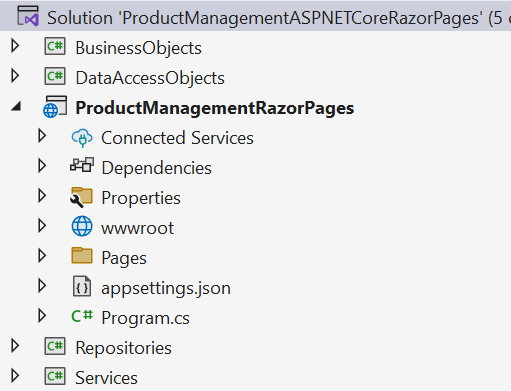
# Activity 01: Build a solution by Visual Studio.NET

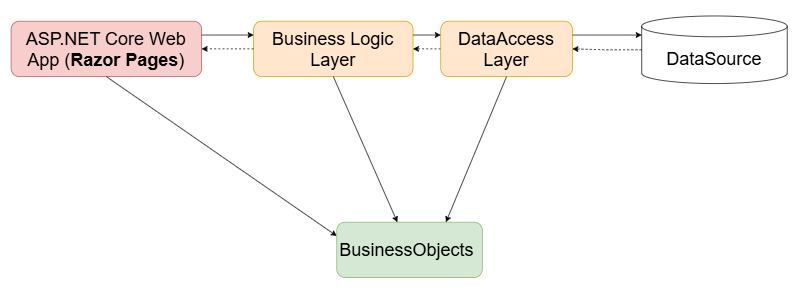
Create a Blank Solution named **ProductManagementASPNETCoreRazorPages** then add new a **Class Library** project named **BusinessObjects, DataAccessObjects, Repositories, Services** and an ASP.NET Core Web App (Razor Pages) project named **ProductManagementRazorPages**

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

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

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



****

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); Database=MyStore; Uid=sa; Pwd=1234567890**” Microsoft.EntityFrameworkCore.SqlServer --output-dir ./

* 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"];*

*}*

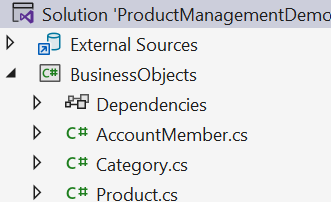
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

        {

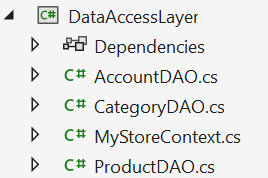
***optionsBuilder.UseSqlServer(GetConnectionString());***

         }

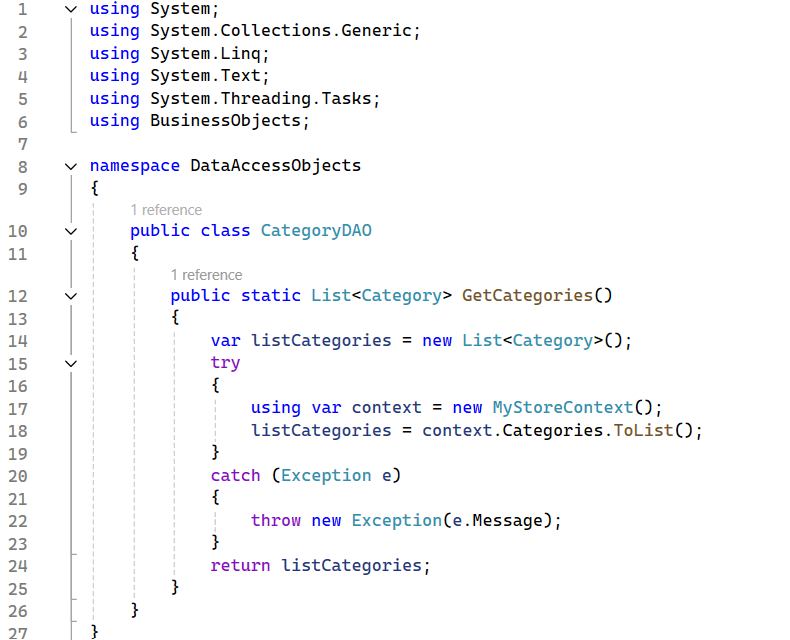
* Move the MyStoreContext.cs to DataAccessLayer Project



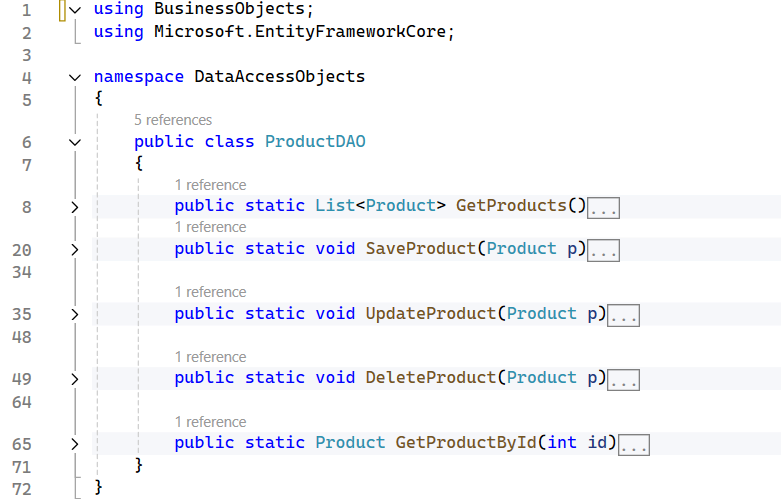
# Activity 03: Write codes for the DataAccessLayer project



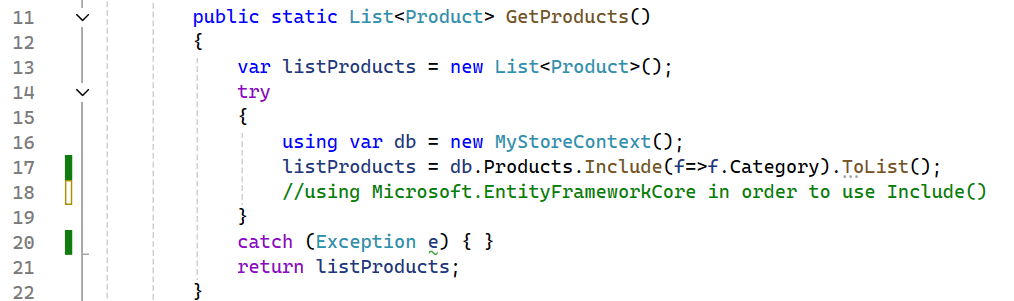
## **Step 01.** On the **DataAccessObjects** project, add a class named **CategoryDAO.cs** and write codes as follows:

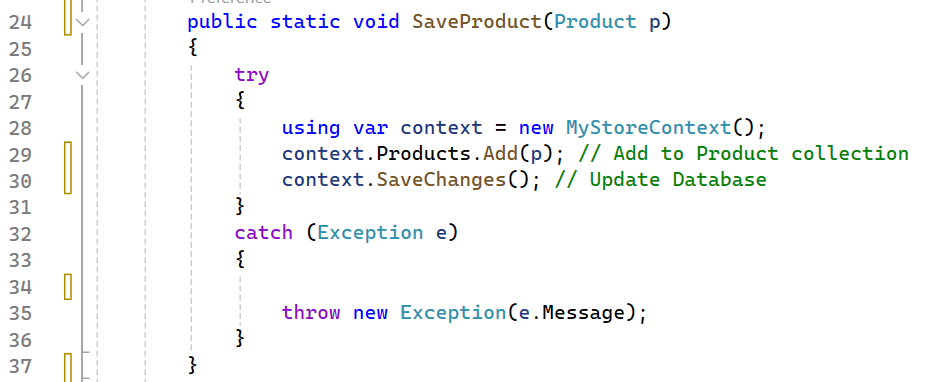


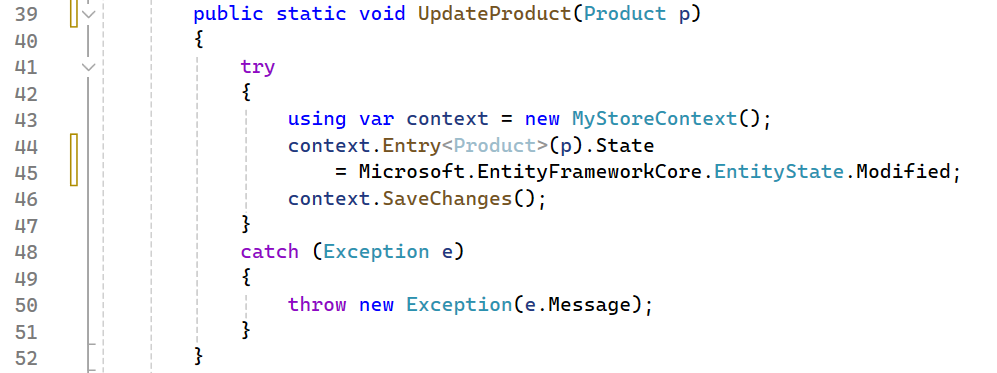
## **Step 02**. On the **DataAccessObjects** project, add a class named **ProductDAO.cs** and write codes as follows:

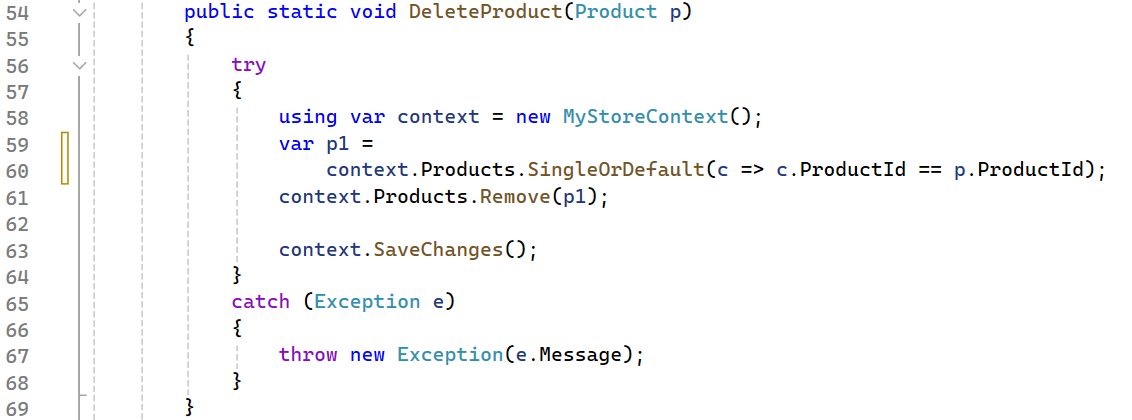


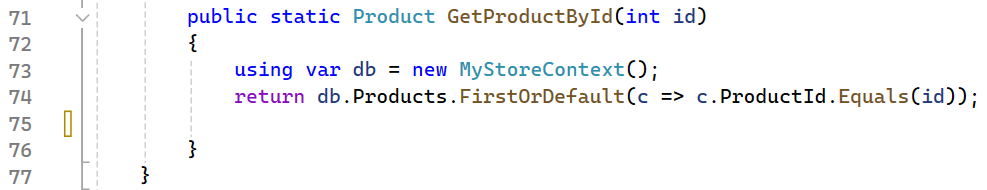
The details of functions in ProductDAO.cs:



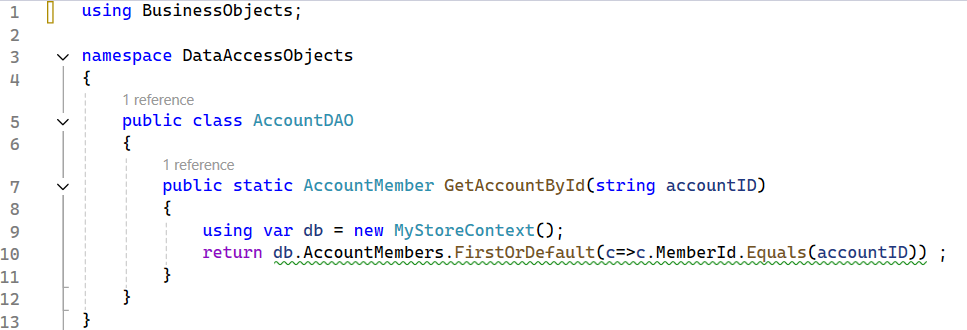




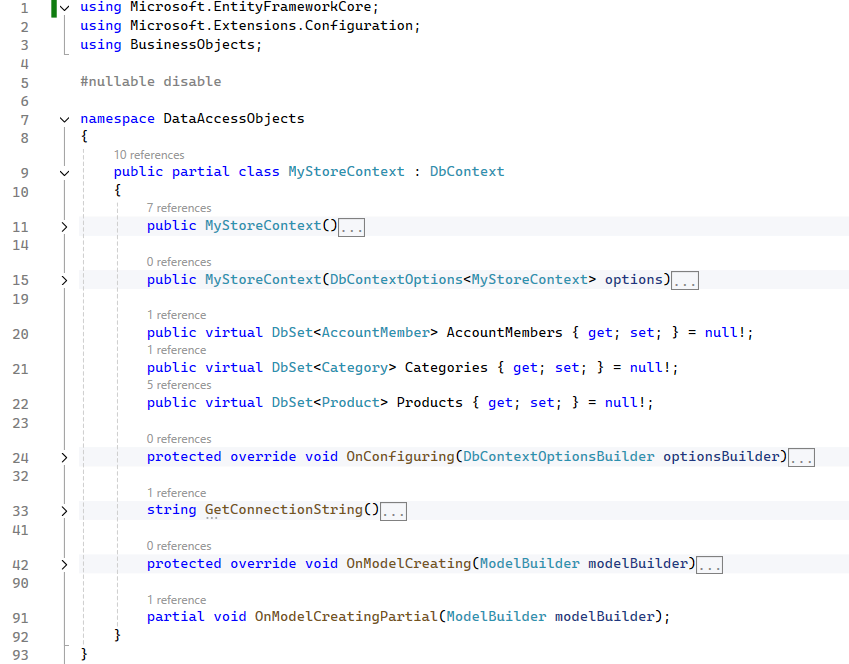




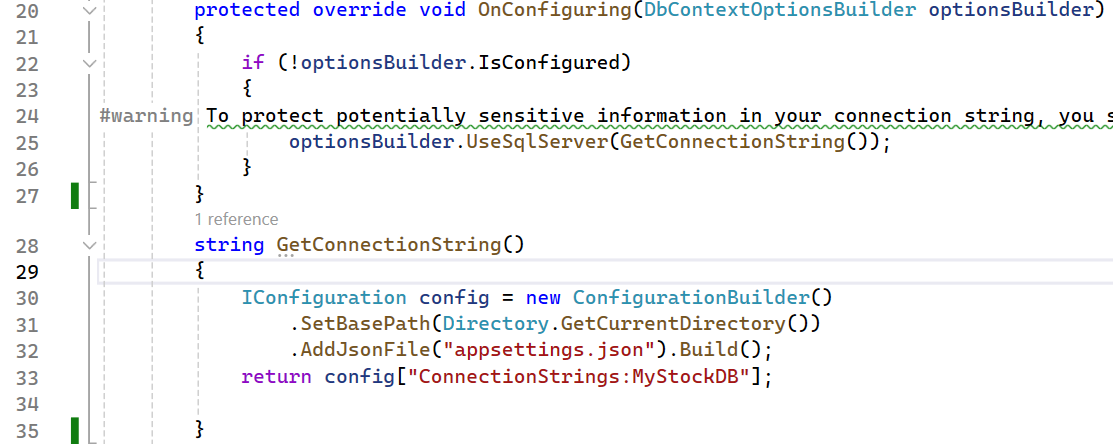
## **Step 03**. Write codes for **AccountDAO.cs** as follows:



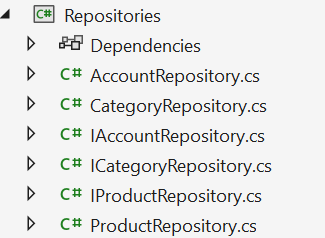
**Step 04**. The codes for **MyStoreContext.cs**:



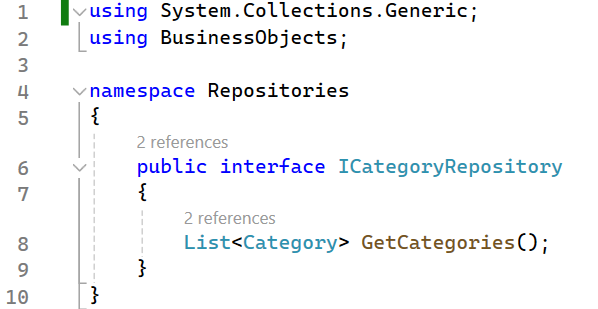
The details for GetConnectionString() and OnConfiguring() functions



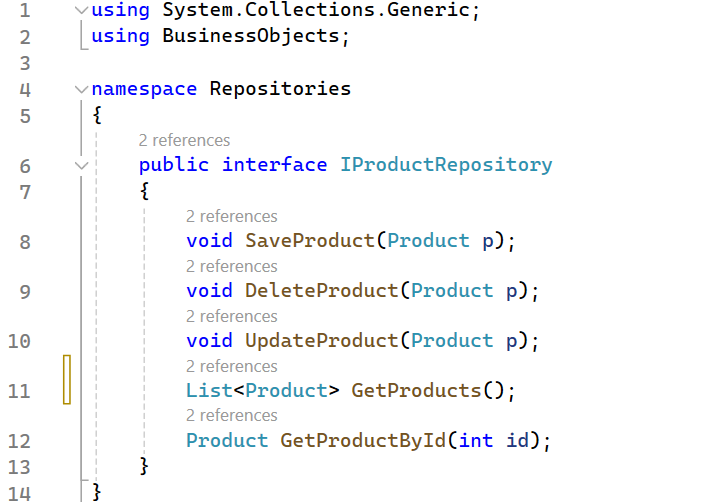
# Activity 04: Write codes for the Repositories project



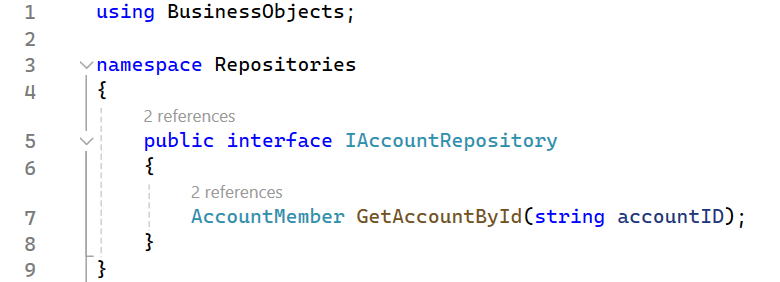
## **Step 01.** On the **Repositories** project, add an interface named **ICatergoryRepository.cs** and write codes as follows:



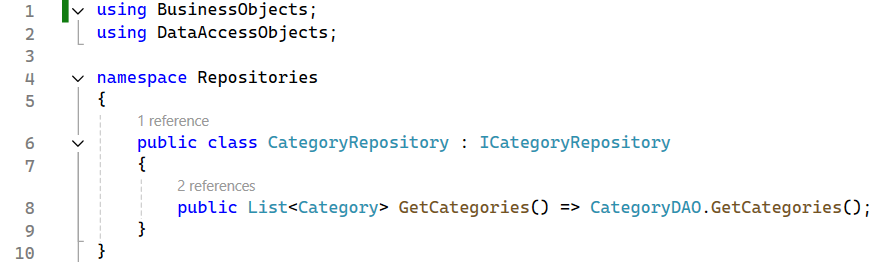
## **Step 02**. On the **Repositories** project, add an interface named **IProductRepository.cs** and write codes as follows:



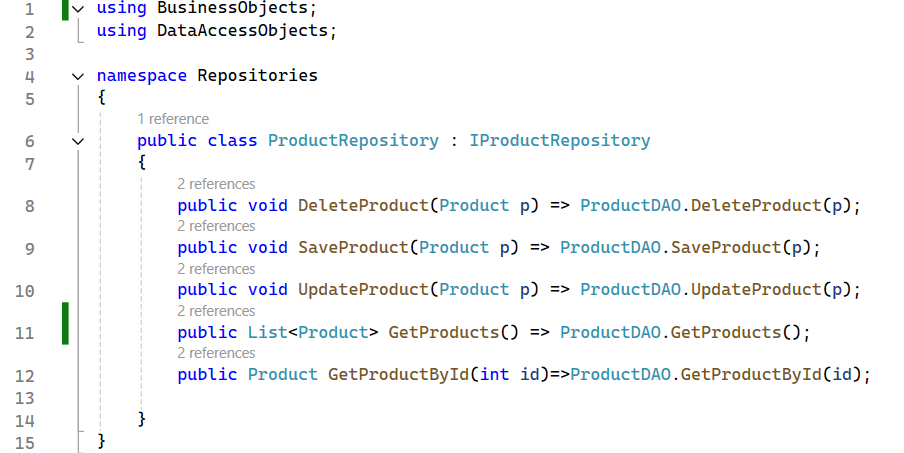
## **Step 03**. On the **Repositories** project, add an interface named **IAccountRepository.cs** and write codes as follows:



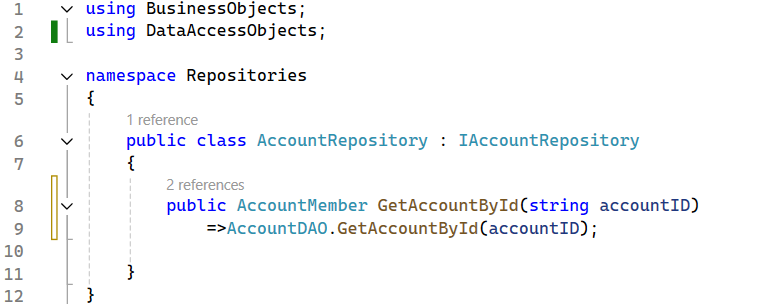
**Step 04**. Write codes for class **CategoryRepository.cs** as follows:



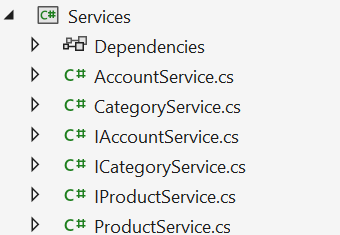
**Step 05**. Write codes for class **ProductRepository.cs** as follows:



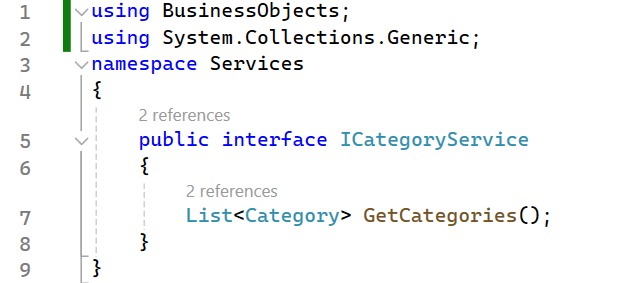
**Step 06**. Write codes for class **AccountRepository.cs** as follows:



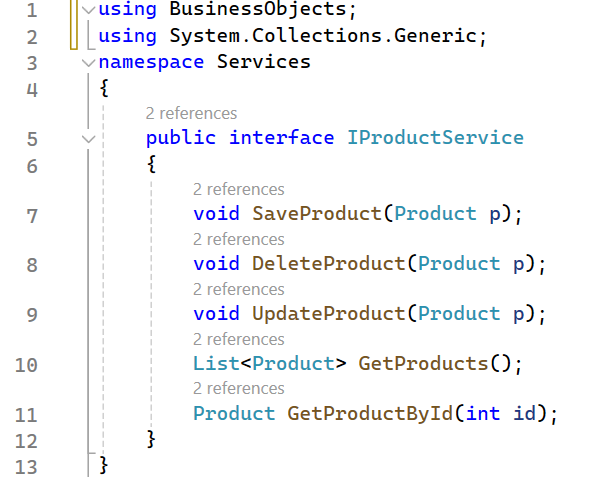
# Activity 05: Write codes for the Services project



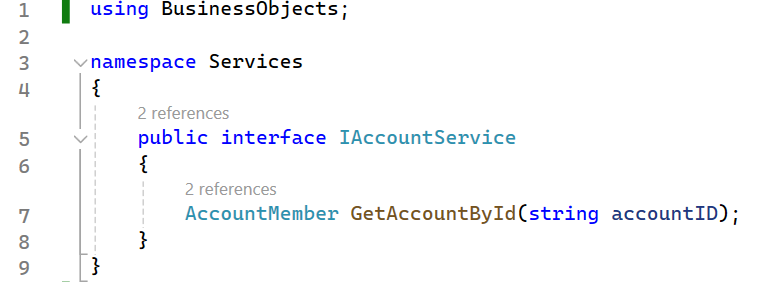
## **Step 01.** On the **Services** project, add an interface named **ICatergoryService.cs** and write codes as follows:



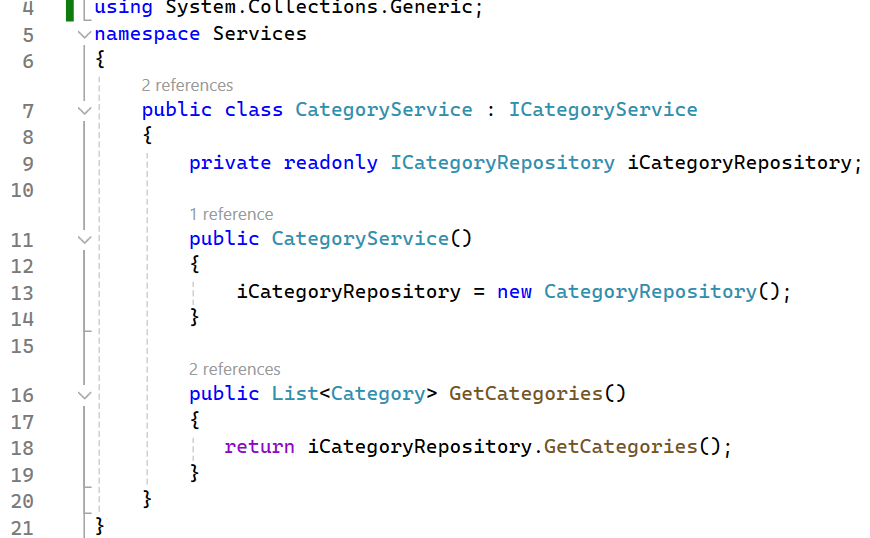
## **Step 02**. On the **Services** project, add an interface named **IProductService.cs** and write codes as follows:



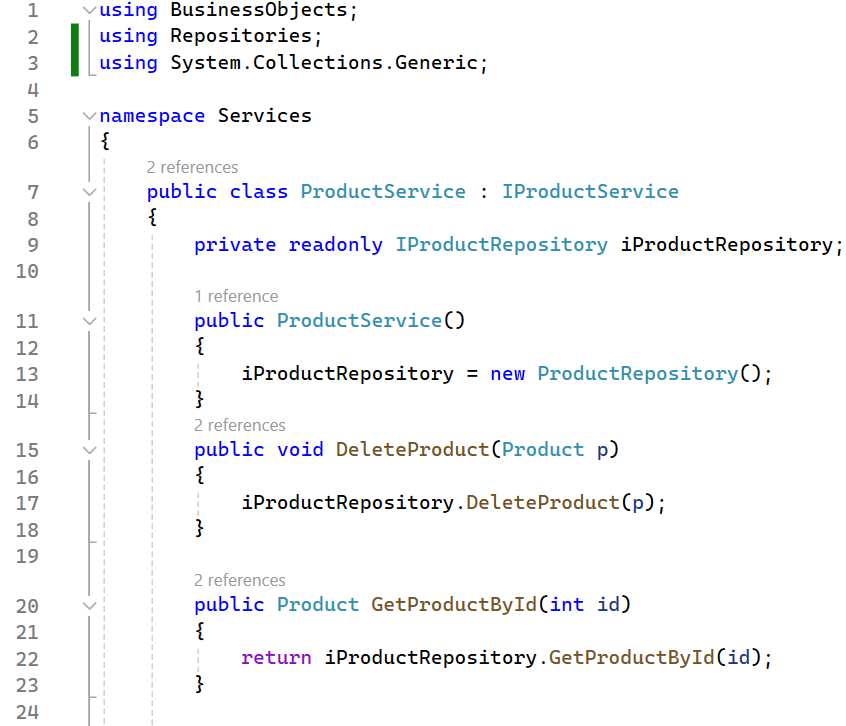
## **Step 03**. On the **Services** project, add an interface named **IAccountService.cs** and write codes as follows:

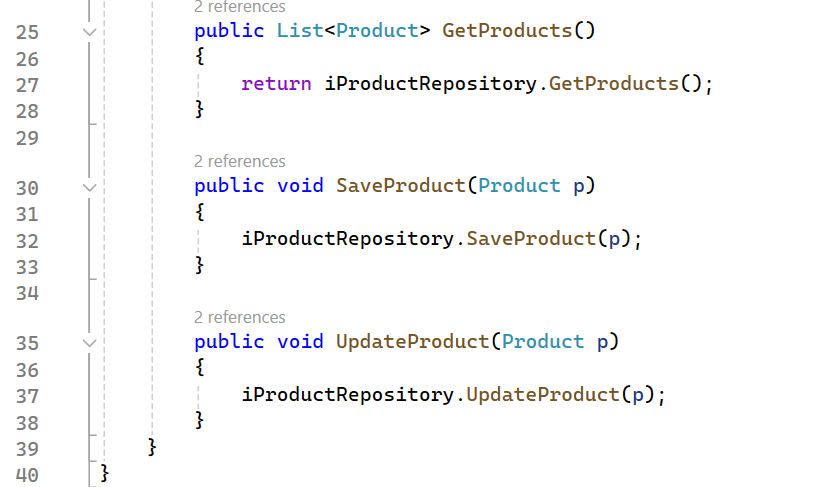


**Step 04**. Write codes for class **CategoryService.cs** as follows:

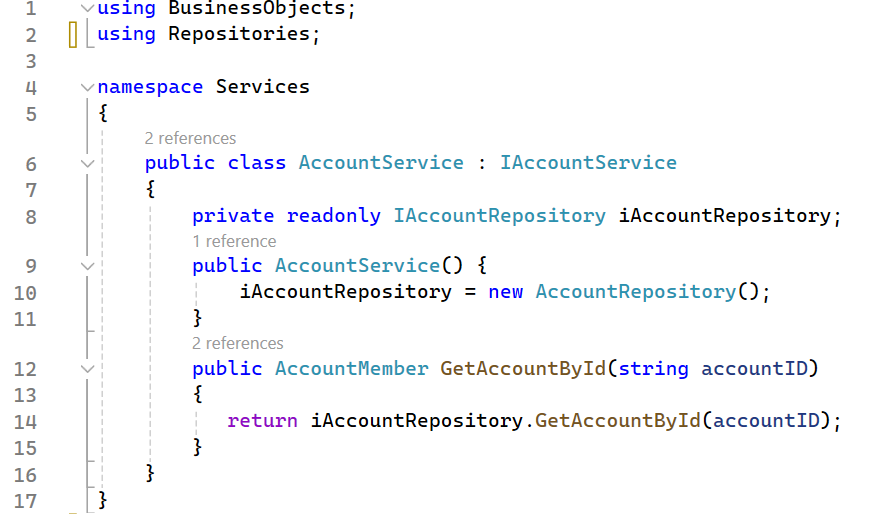


**Step 05**. Write codes for class **ProductService.cs** as follows:





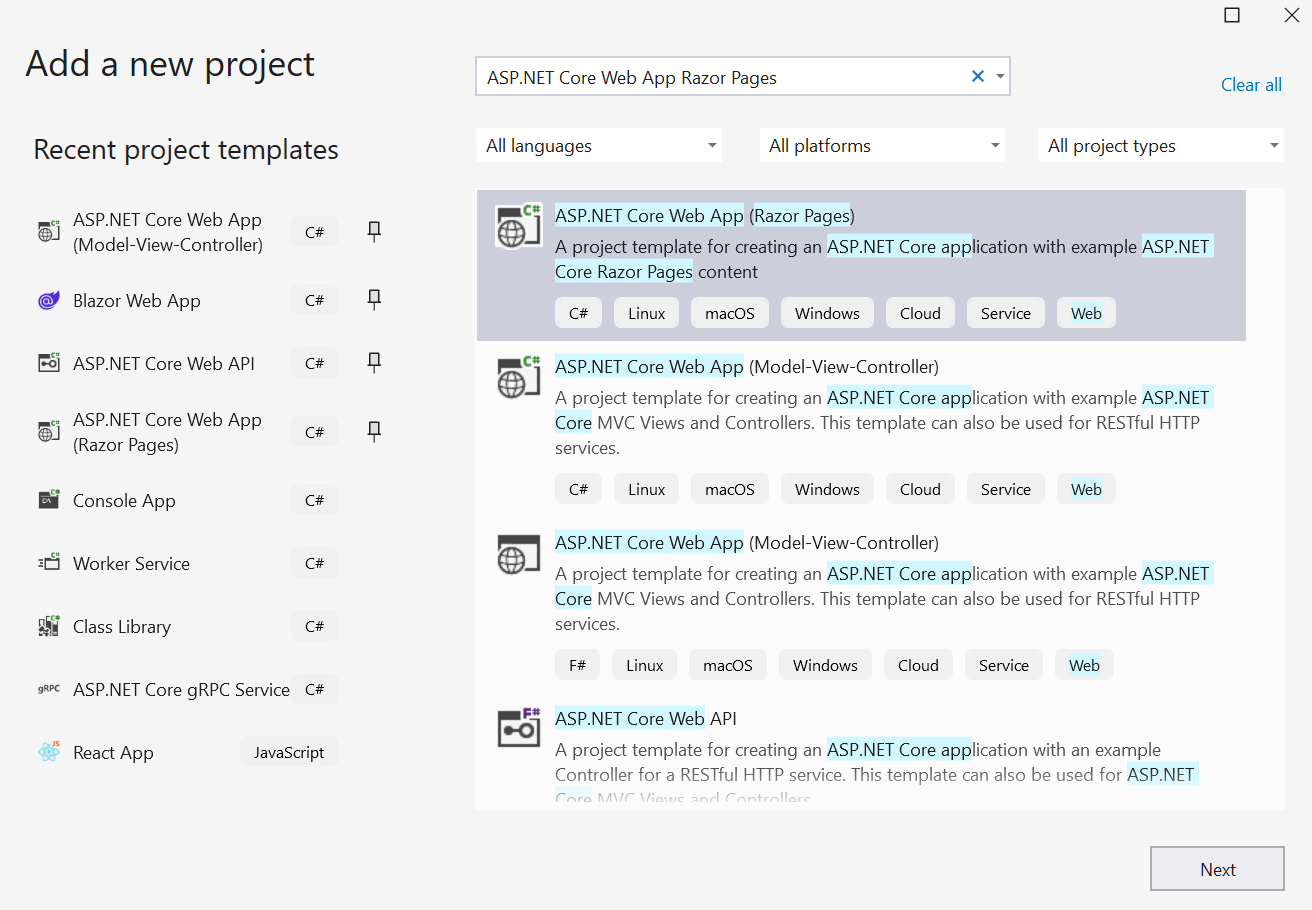
**Step 06**. Write codes for class **AccountService.cs** as follows:

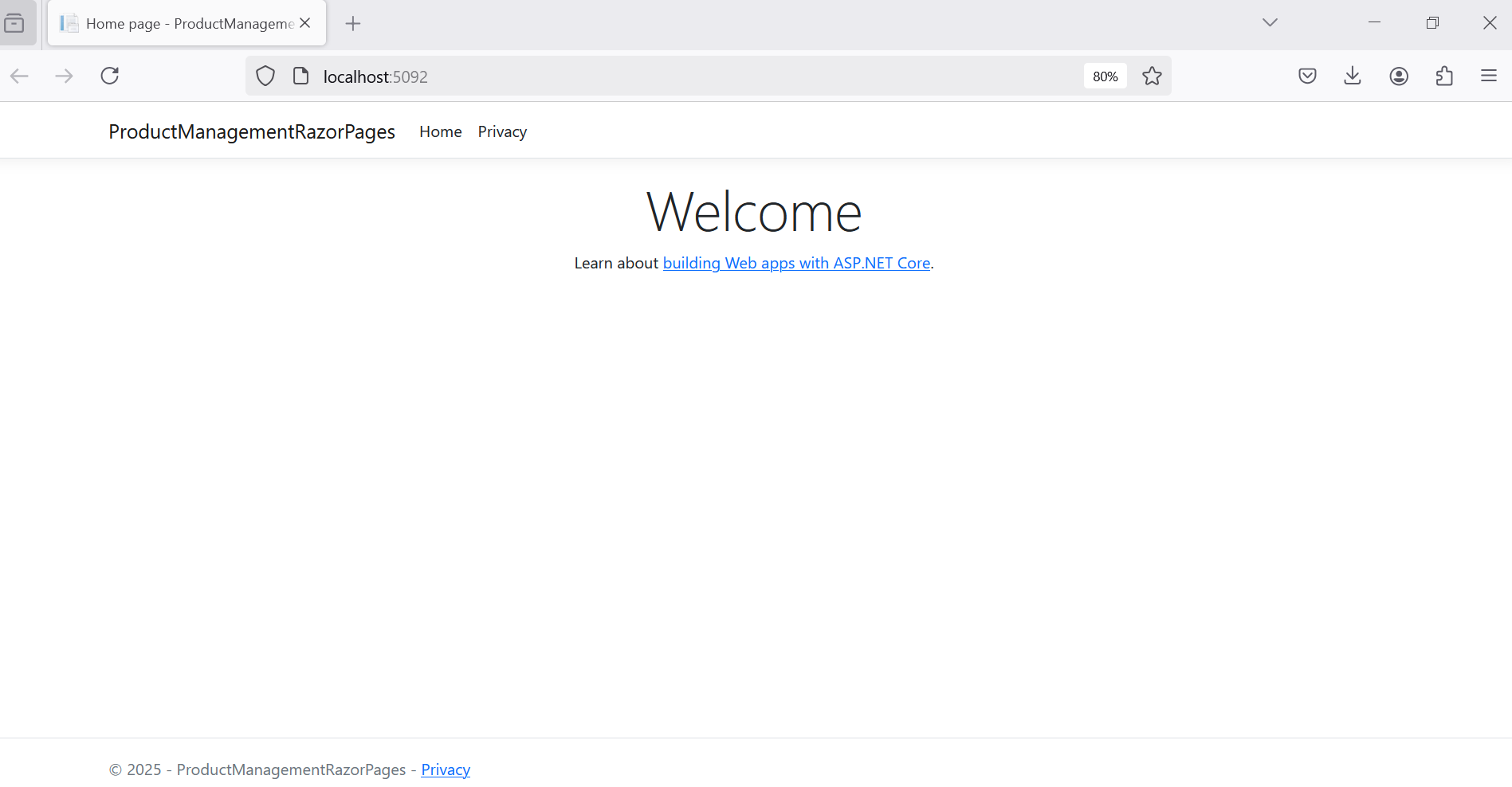


# Activity 06: Work with ASP.NET Core Web App (Razor Pages)

## **Step 01.** Create and run the **ASP.NET Core Web App (Razor Pages)** project, the result as the following

Create a new project type ASP.NET Core Web App (Razor Pages)



****

## **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"

}

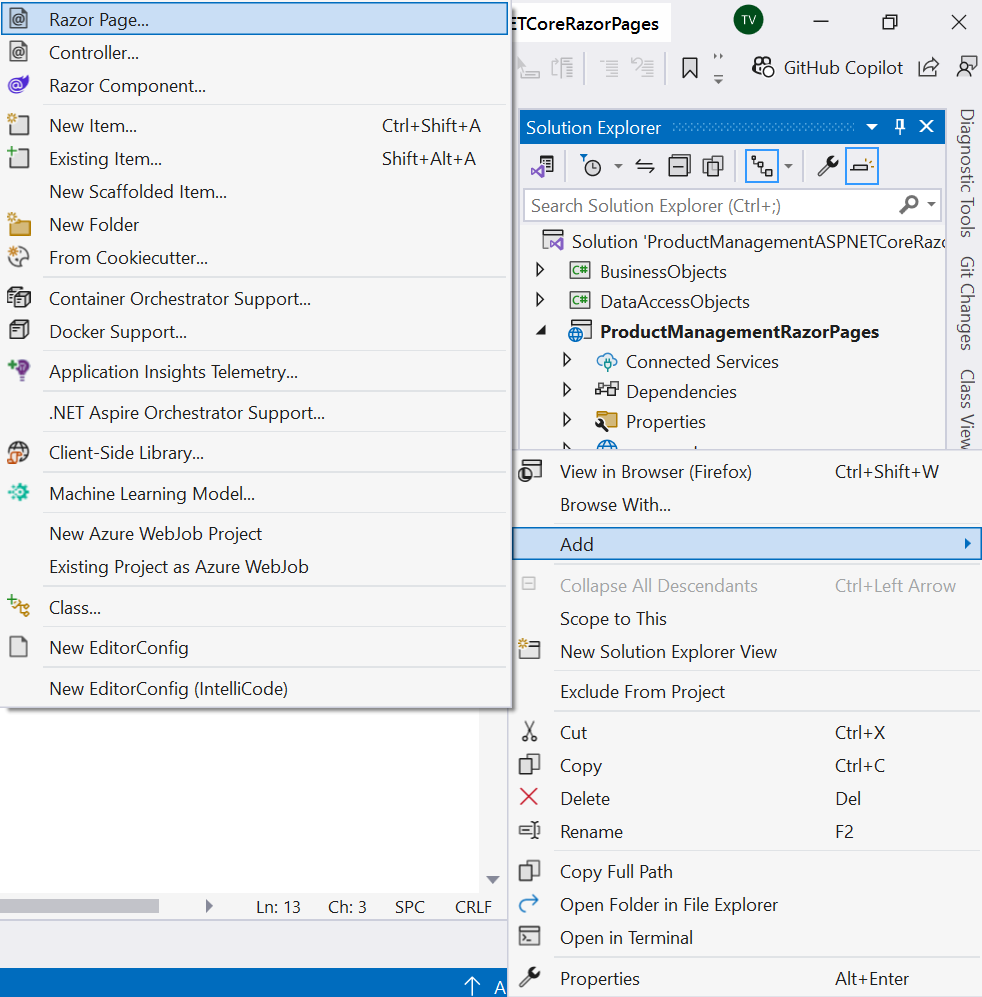
}

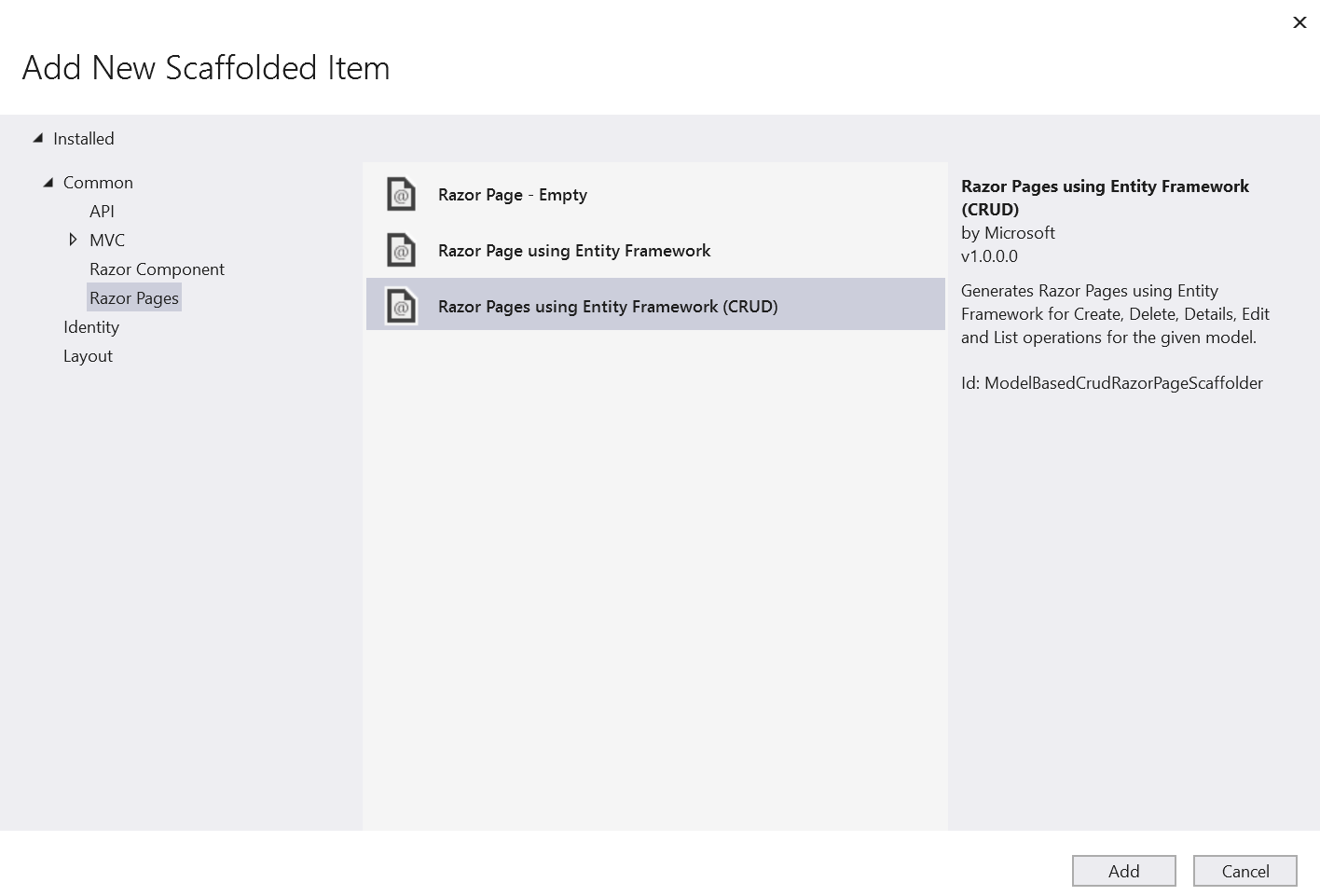
**Step 03.** Add **Business Objects** and **Services** projects as references for the ASP.NET Core Web App (Razor Pages)

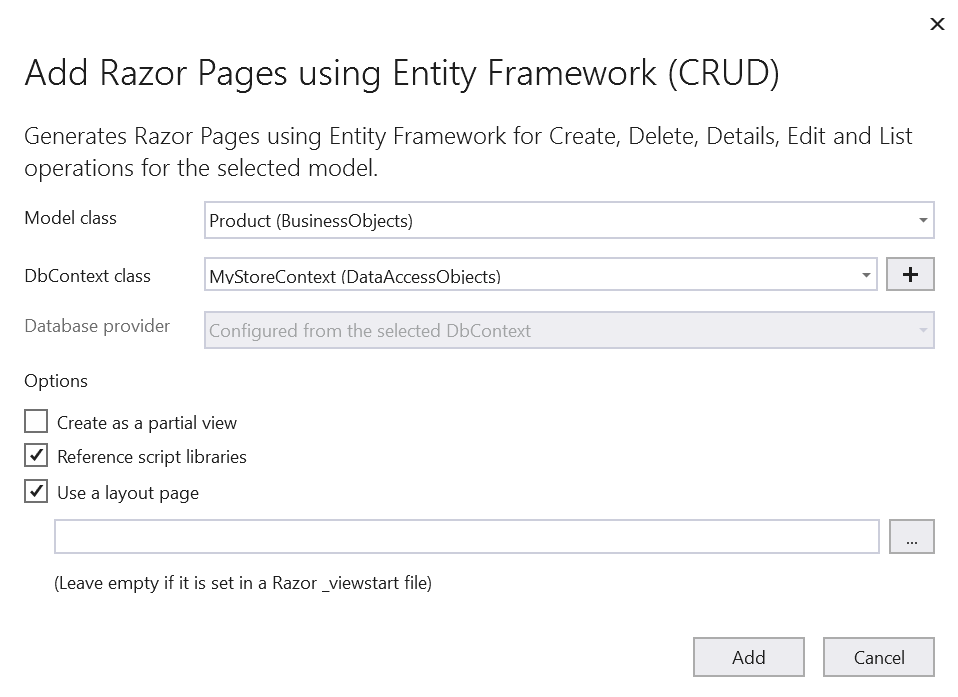
## **Step 04.** Create Razor Pages

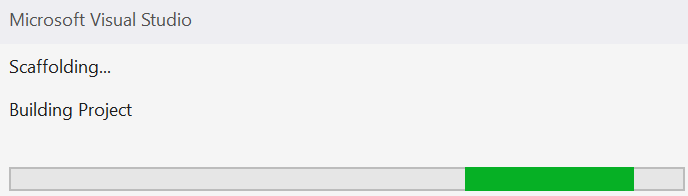
* Connect direct to the Data Access Layer (MyStoreContext.cs) to generate code
* Then
  + Add Dependency Injection
  + Change the code connects to the Service Layer

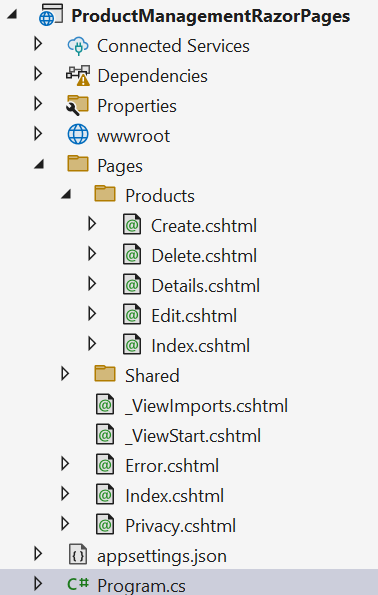
Add new Razor Pages





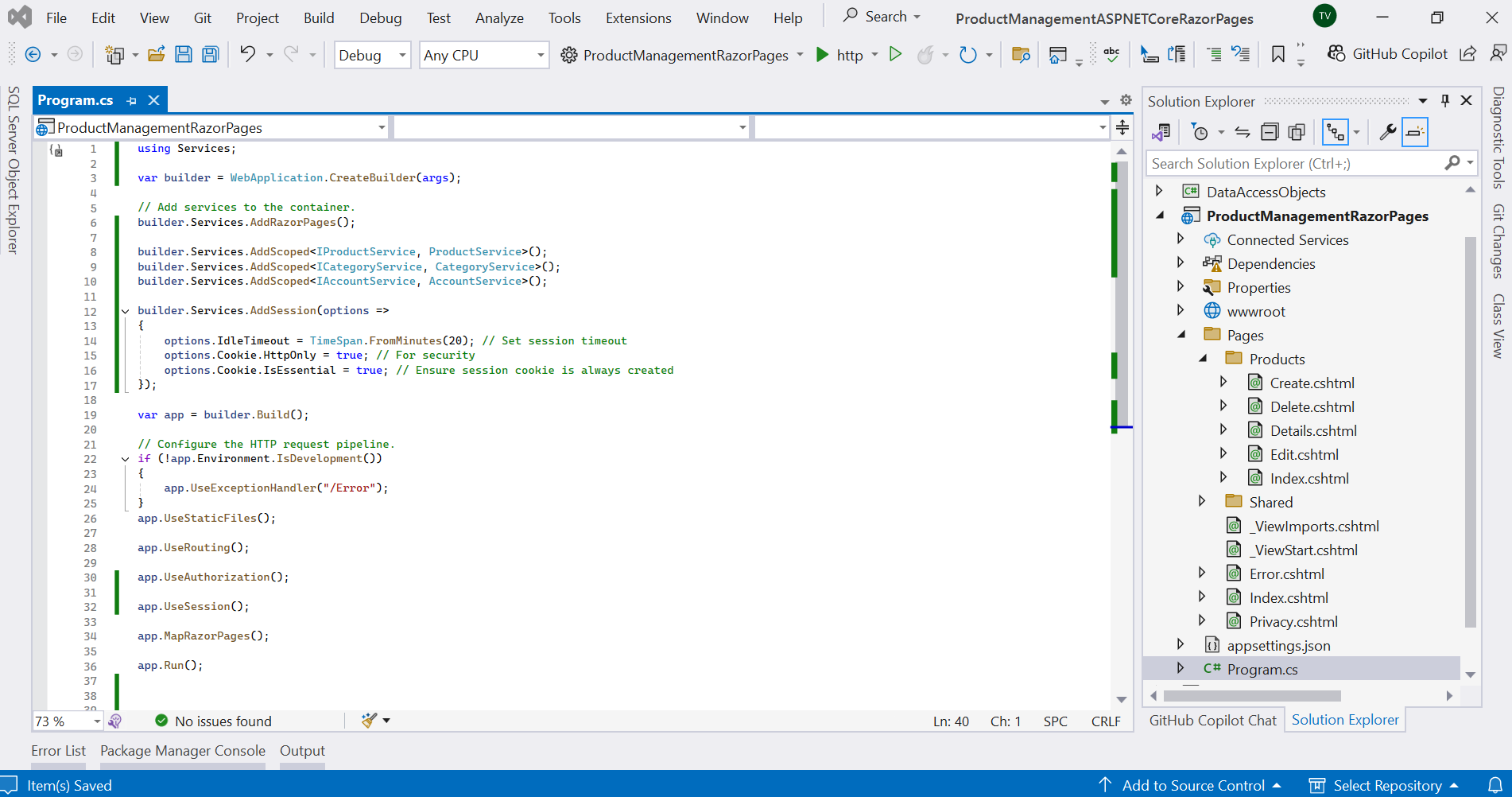




****

## **Step 05.** After generate the Pages/Products/Create.cshtml, …, change the code as the following

***Program.cs***



using Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages();

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

builder.Services.AddScoped<ICategoryService, CategoryService>();

builder.Services.AddScoped<IAccountService, AccountService>();

builder.Services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(20); // Set session timeout

options.Cookie.HttpOnly = true; // For security

options.Cookie.IsEssential = true; // Ensure session cookie is always created

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseSession();

app.MapRazorPages();

app.Run();

*Use Dependency Injection for IProductService, ICategoryService*

*Code behind of Razor Pages for Create function*

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.AspNetCore.Mvc.Rendering;

using BusinessObjects;

using Services;

namespace ProductManagementRazorPages.Pages.Products

{

public class CreateModel : PageModel

{

private readonly IProductService \_contextProduct;

private readonly ICategoryService \_contextCategory;

public CreateModel(IProductService context, ICategoryService categoryService)

{

\_contextProduct = context;

\_contextCategory = categoryService;

}

public IActionResult OnGet()

{

ViewData["CategoryId"] = new SelectList(\_contextCategory.GetCategories(), "CategoryId", "CategoryId");

return Page();

}

[BindProperty]

public Product Product { get; set; } = default!;

// For more information, see https://aka.ms/RazorPagesCRUD.

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

\_contextProduct.SaveProduct(Product);

return RedirectToPage("./Index");

}

}

}

*Code behind of Razor Pages for Edit function*

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.AspNetCore.Mvc.Rendering;

using Microsoft.EntityFrameworkCore;

using BusinessObjects;

using Services;

namespace ProductManagementRazorPages.Pages.Products

{

public class EditModel : PageModel

{

private readonly IProductService \_context;

private readonly ICategoryService \_categoryService;

public EditModel(IProductService context, ICategoryService categoryService)

{

\_context = context;

\_categoryService = categoryService;

}

[BindProperty]

public Product Product { get; set; } = default!;

public async Task<IActionResult> OnGetAsync(int? id)

{

if (id == null)

{

return NotFound();

}

var product = \_context.GetProductById((int)id);

if (product == null)

{

return NotFound();

}

Product = product;

ViewData["CategoryId"] = new SelectList(\_categoryService.GetCategories(), "CategoryId", "CategoryId");

return Page();

}

// To protect from overposting attacks, enable the specific properties you want to bind to.

// For more information, see https://aka.ms/RazorPagesCRUD.

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

try

{

\_context.UpdateProduct(Product);

}

catch (DbUpdateConcurrencyException)

{

if (!ProductExists(Product.ProductId))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToPage("./Index");

}

private bool ProductExists(int id)

{

return (\_context.GetProductById(id) == null) ? true : false;

}

}

}

*Code behind of Razor Pages for List all items function*

using Microsoft.AspNetCore.Mvc.RazorPages;

using BusinessObjects;

using Services;

namespace ProductManagementRazorPages.Pages.Products

{

public class IndexModel : PageModel

{

private readonly IProductService \_contextProduct;

public IndexModel(IProductService context)

{

\_contextProduct = context;

}

public IList<Product> Product { get;set; } = default!;

public async Task OnGetAsync()

{

Product = \_contextProduct.GetProducts();

}

}

}

*Code behind of Razor Pages for Delete function*

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using BusinessObjects;

using Services;

namespace ProductManagementRazorPages.Pages.Products

{

public class DeleteModel : PageModel

{

private readonly IProductService \_context;

public DeleteModel(IProductService context)

{

\_context = context;

}

[BindProperty]

public Product Product { get; set; } = default!;

public async Task<IActionResult> OnGetAsync(int? id)

{

if (id == null)

{

return NotFound();

}

var product = \_context.GetProductById ((int) id);

if (product == null)

{

return NotFound();

}

else

{

Product = product;

}

return Page();

}

public async Task<IActionResult> OnPostAsync(int? id)

{

if (id == null)

{

return NotFound();

}

var product = \_context.GetProductById((int) id);

if (product != null)

{

Product = product;

\_context.DeleteProduct(product);

}

return RedirectToPage("./Index");

}

}

}

*Code behind of Razor Pages for Details function*

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.EntityFrameworkCore;

using BusinessObjects;

using Services;

namespace ProductManagementRazorPages.Pages.Products

{

public class DetailsModel : PageModel

{

private readonly IProductService \_context;

public DetailsModel(IProductService context)

{

\_context = context;

}

public Product Product { get; set; } = default!;

public async Task<IActionResult> OnGetAsync(int? id)

{

if (id == null)

{

return NotFound();

}

var product = \_context.GetProductById((int) id);

if (product == null)

{

return NotFound();

}

else

{

Product = product;

}

return Page();

}

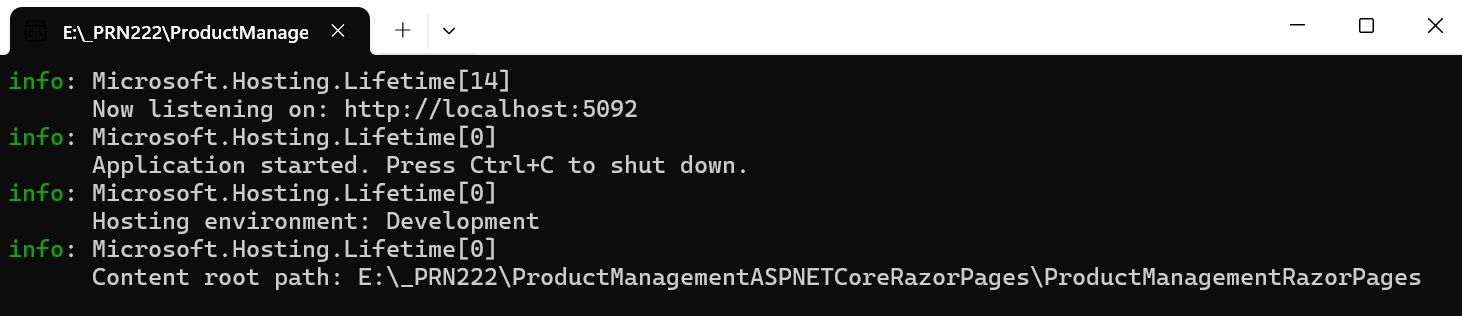
}

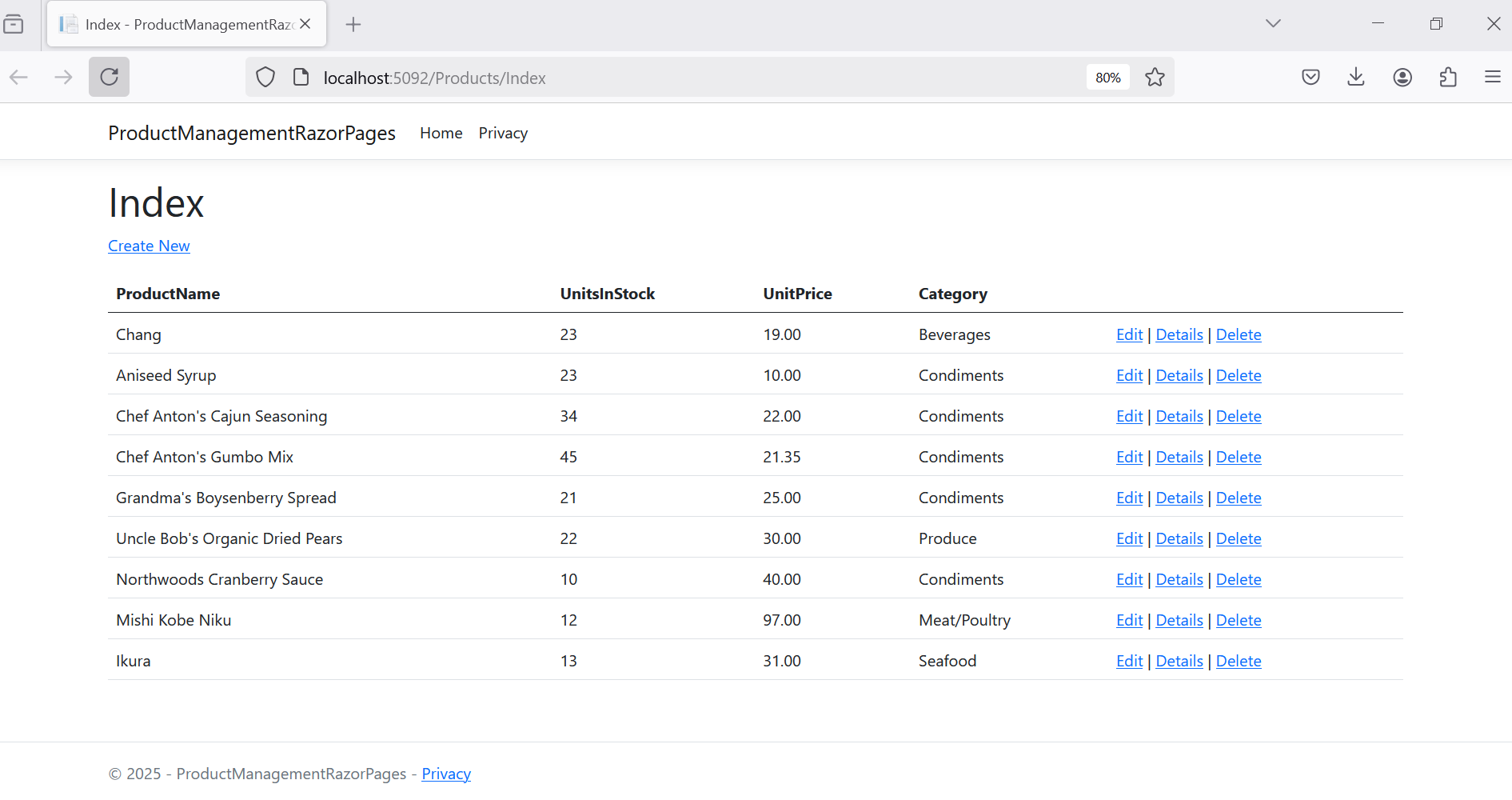
}

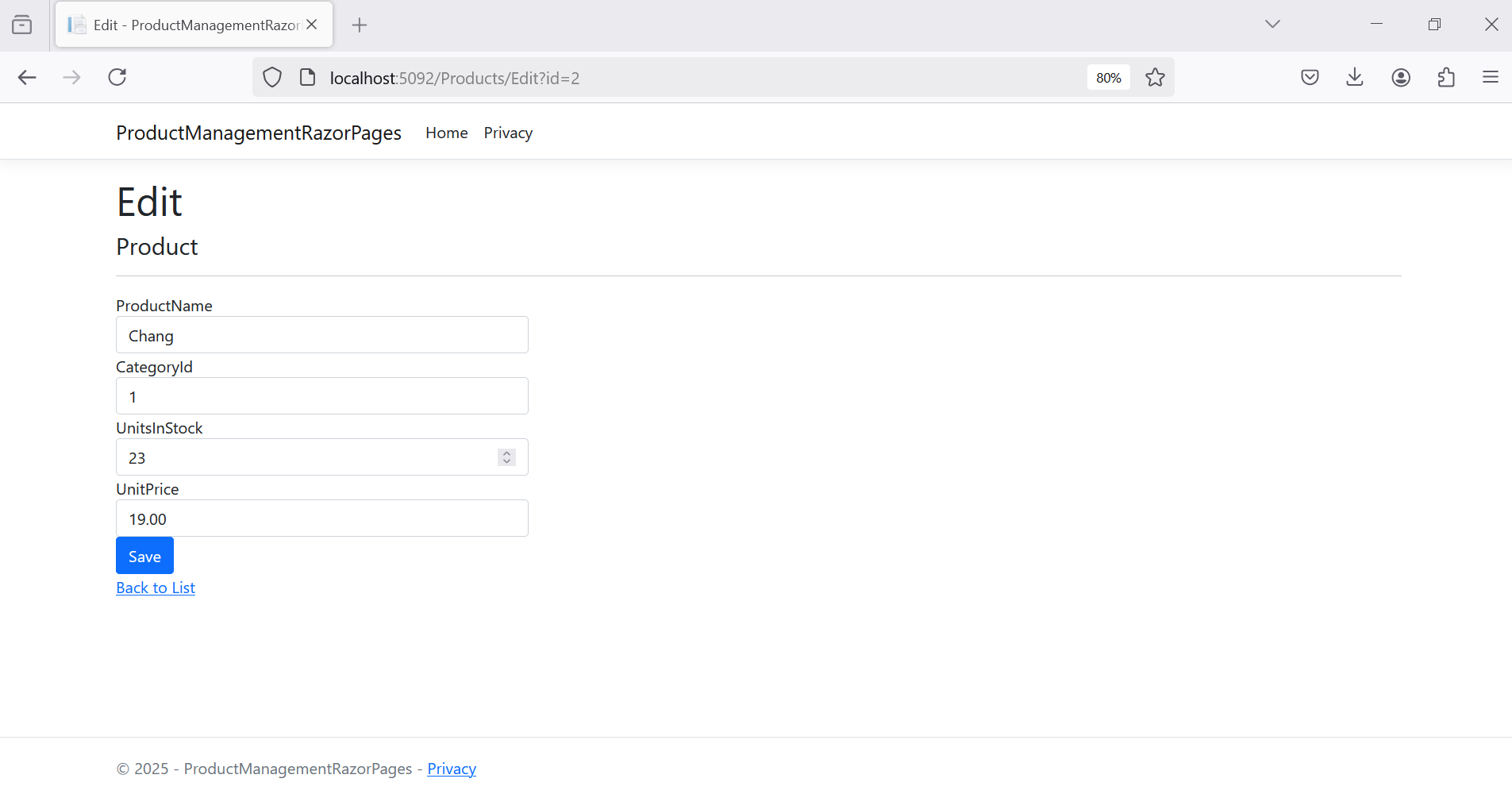
## **Step 06.** Change the View in the Razor View depend on your template.

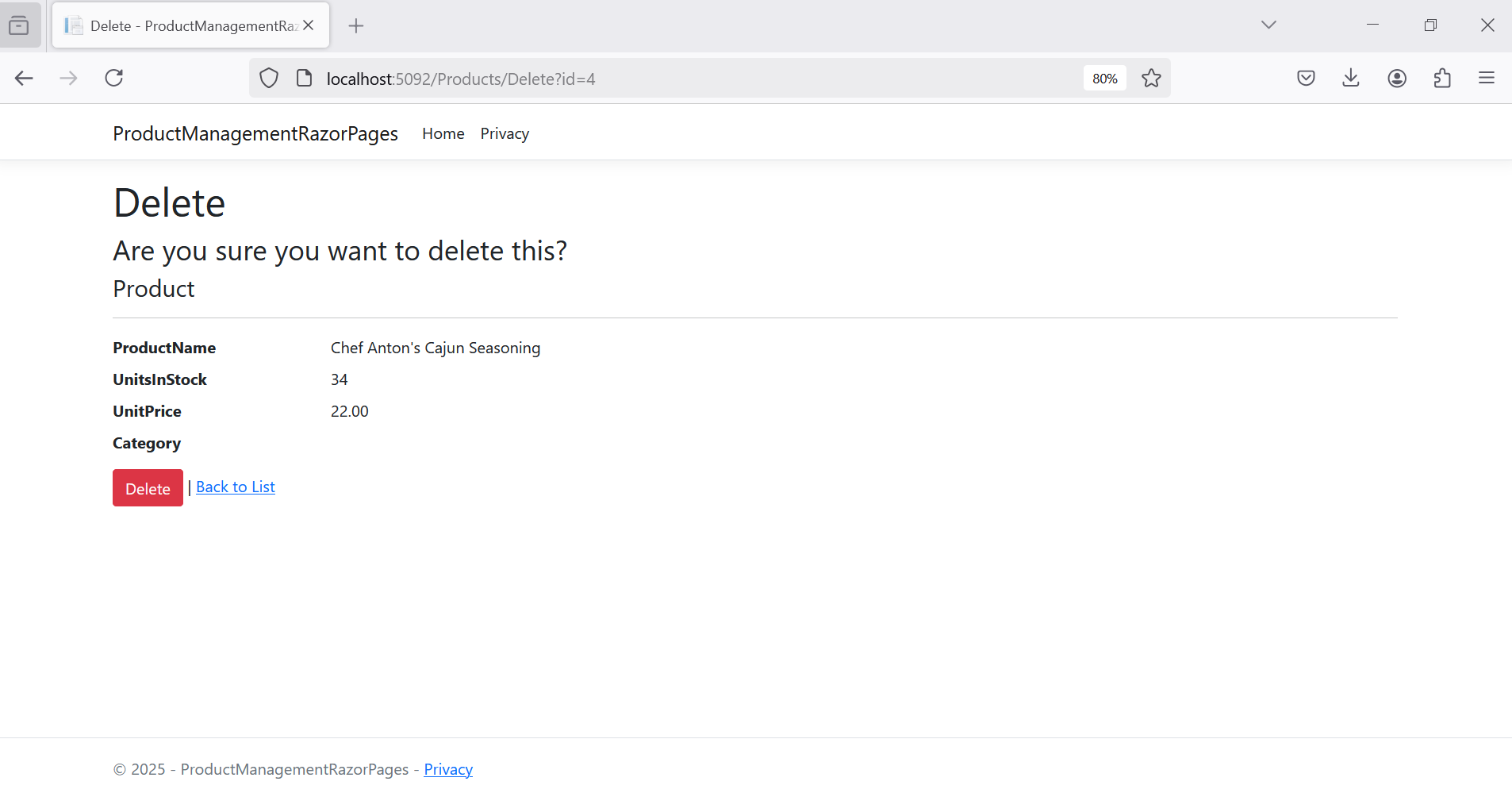
## **Step 07.** Clean ASP.NET Core Web App (Razor Pages) project, remove Entity Framework Core related packages.

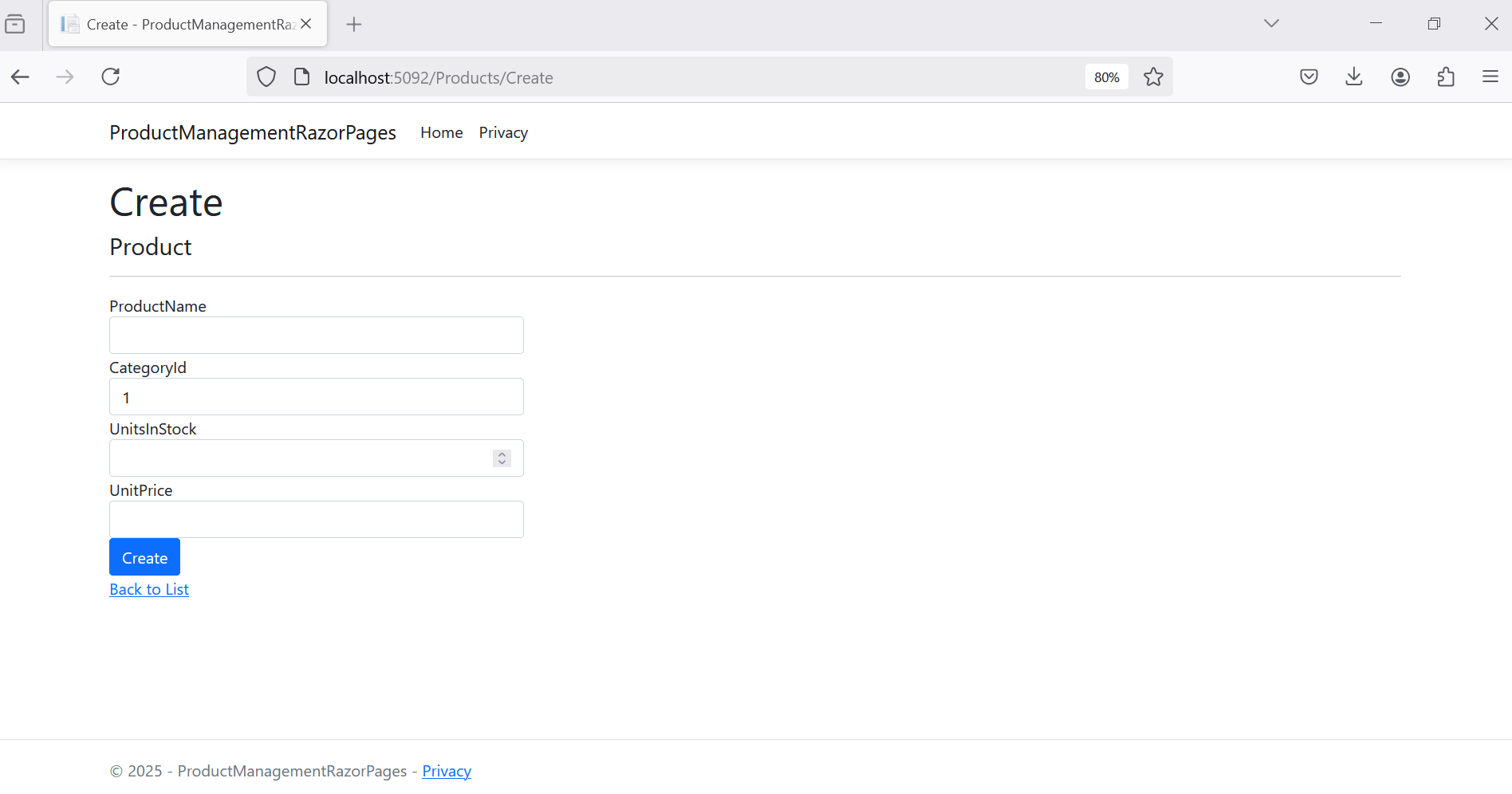
# Activity 07: Run the ASP.NET Core Web App (Razor Pages) project and test all actions











# Activity 08: Session Management with ASP.NET Core Web App (Razor Pages)

## **Step 01.** Add the Session service in IoC Container of Kestrel Server, and then add the Middleware in HTTP request pipeline in Program.cs

using Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages();

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

builder.Services.AddScoped<ICategoryService, CategoryService>();

builder.Services.AddScoped<IAccountService, AccountService>();

builder.Services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(20); // Set session timeout

options.Cookie.HttpOnly = true; // For security

options.Cookie.IsEssential = true; // Ensure session cookie is always created

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseSession();

app.UseAuthorization();

app.MapRazorPages();

app.Run();

## **Step 02.** Create the Login Page (In this case, using MemberId and MemberPassword for authentication process.

*Pages/Login.cshtml*

@page

@model ProductManagementRazorPages.Pages.LoginModel

@{

ViewData["Title"] = "Login";

}

<h1>Login</h1>

<h4>AccountMember</h4>

<hr />

<div class="row">

<div class="col-md-4">

<**form** method="post">

<**div** **asp-validation-summary**="ModelOnly" class="text-danger"></**div**>

<div class="form-group">

<**label** **asp-for**="AccountMember.MemberId" class="control-label"></**label**>

<**input** **asp-for**="AccountMember.MemberId" class="form-control" />

<**span** **asp-validation-for**="AccountMember.MemberId" class="text-danger"></**span**>

</div>

<div class="form-group">

<**label** **asp-for**="AccountMember.MemberPassword" class="control-label"></**label**>

<**input** **type**="password" **asp-for**="AccountMember.MemberPassword" class="form-control" />

<**span** **asp-validation-for**="AccountMember.MemberPassword" class="text-danger"></**span**>

</div>

<div class="form-group">

<input type="submit" value="Create" class="btn btn-primary" />

</div>

</**form**>

</div>

</div>

<div>

<**a** **asp-page**="Index">Back to List</**a**>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

*Pages/Login.cshtml.cs*

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using BusinessObjects;

using Services;

namespace ProductManagementRazorPages.Pages

{

public class LoginModel : PageModel

{

private IAccountService \_accountService; // using Dependency Injection

public LoginModel(IAccountService accountService)

{

\_accountService = accountService;

}

public async Task<IActionResult> OnGetAsync()

{

var loginId = HttpContext.Session.GetInt32("Account").ToString();

if (!string.IsNullOrEmpty(loginId))

{

return RedirectToPage("/Products/Index");

}

return Page();

}

[BindProperty]

public AccountMember AccountMember { get; set; } = default!;

public string ErrorMessage { get; set; }

// To protect from overposting attacks, see https://aka.ms/RazorPagesCRUD

public async Task<IActionResult> OnPostAsync()

{

var loginId = HttpContext.Session.GetInt32("Account").ToString();

if (!string.IsNullOrEmpty(loginId))

{

return RedirectToPage("/Products/Index");

}

var memberAccount = \_accountService.GetAccountById(AccountMember.MemberId);

if (memberAccount == null)

{

ErrorMessage = "You do not have permission to do this function!";

ModelState.AddModelError(string.Empty, ErrorMessage);

return Page();

}

else if (memberAccount.MemberRole == 1 || memberAccount.MemberRole == 2)

{

HttpContext.Session.SetInt32("Account", memberAccount.MemberRole ?? 0);

return RedirectToPage("/Products/Index");

}

else

{

ErrorMessage = "You do not have permission to do this function!";

ModelState.AddModelError(string.Empty, ErrorMessage);

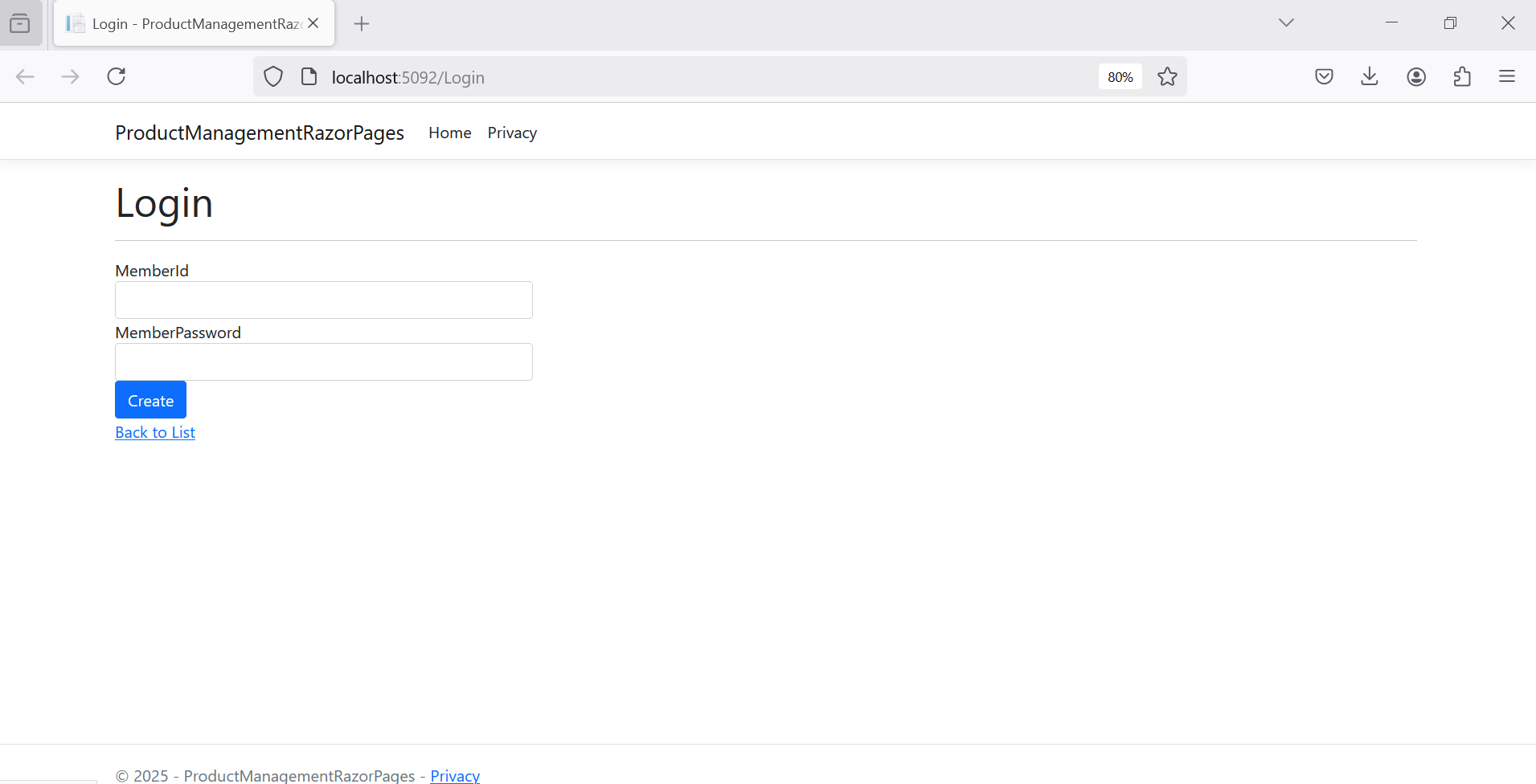
return Page();

}

}

}

}

**

## **Step 03.** Accessing Session data in the Razor Pages.

*string userId = HttpContext.Session.GetString("Accounr");*

*All code CRUD must be changed to check the session is existing or not.*

*Pages/Products/Index.cshtml.cs*

using Microsoft.AspNetCore.Mvc.RazorPages;

using BusinessObjects;

using Services;

using Microsoft.AspNetCore.Mvc;

namespace ProductManagementRazorPages.Pages.Products

{

public class IndexModel : PageModel

{

private readonly IProductService \_contextProduct;

public IndexModel(IProductService context)

{

\_contextProduct = context;

}

public IList<Product> Product { get;set; } = default!;

public async Task<IActionResult> OnGetAsync()

{

if (!string.IsNullOrEmpty(HttpContext.Session.GetString("Account")))

{

Product = \_contextProduct.GetProducts();

return Page();

}

return RedirectToPage("/Login");

}

}

}

# Activity 09: Working SignalR

## **Step 01.** Create Hub class

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.AspNetCore.SignalR;

namespace SignalRLab

{

public class SignalrServer:Hub

{

}

}

## **Step 02.** Add the SignalR service in IoC Container of Kestrel Server, and then add the Middleware in HTTP request pipeline in Program.cs

builder.Services.AddSignalR();

…

app.MapHub<SignalRServer>("/signalRServer");

***Program.cs***

using ProductManagementRazorPages;

using Services;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddRazorPages();

builder.Services.AddSignalR();

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

builder.Services.AddScoped<ICategoryService, CategoryService>();

builder.Services.AddScoped<IAccountService, AccountService>();

builder.Services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(20); // Set session timeout

options.Cookie.HttpOnly = true; // For security

options.Cookie.IsEssential = true; // Ensure session cookie is always created

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.UseSession();

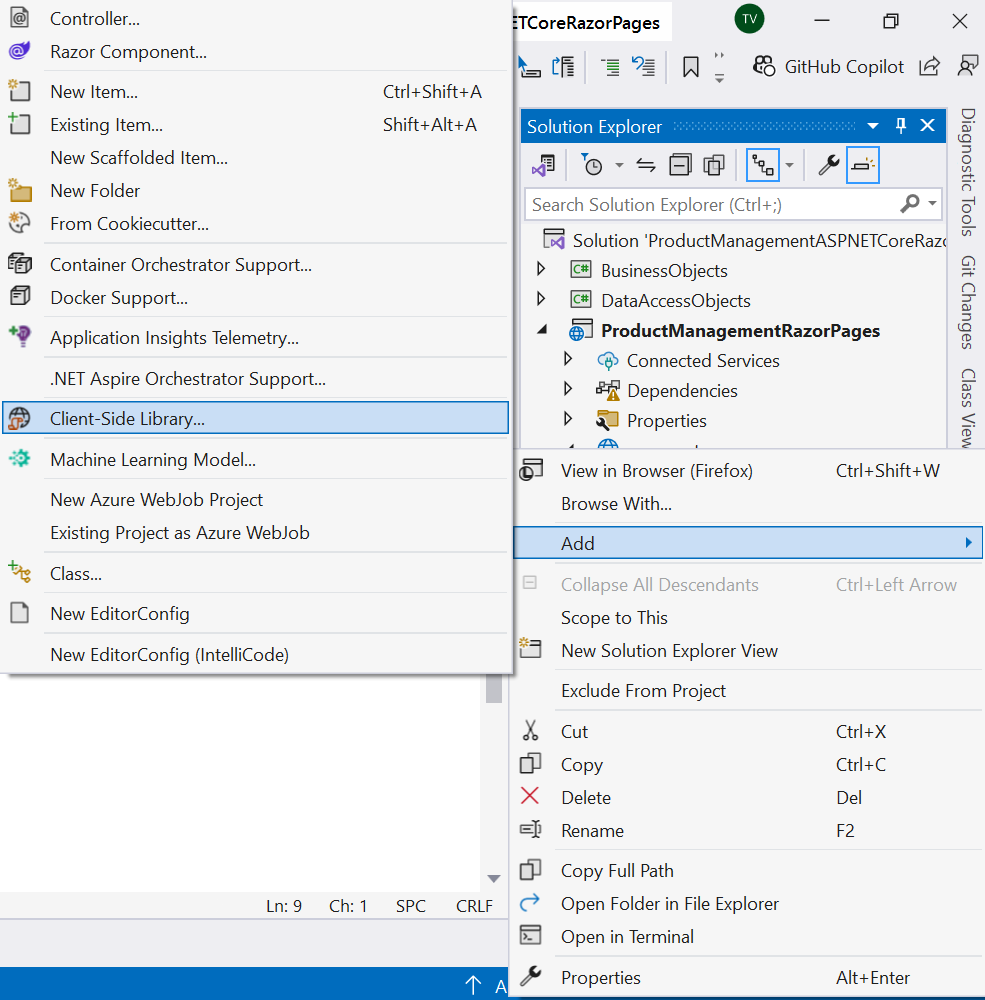
app.MapHub<SignalRServer>("/signalRServer");

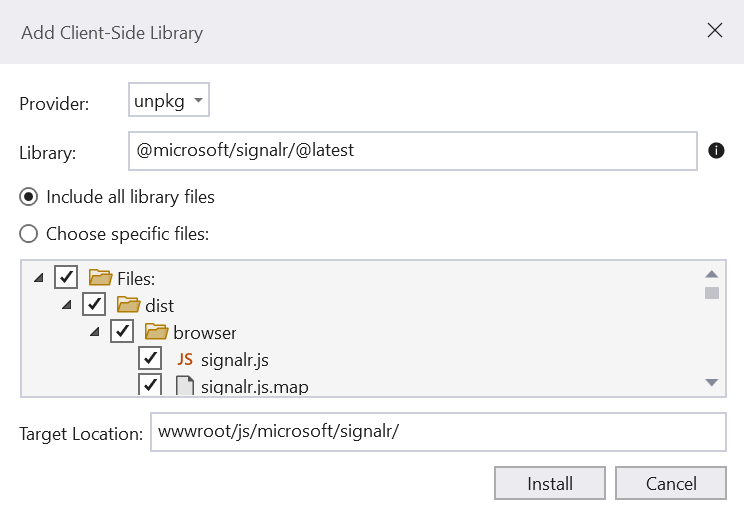
app.MapRazorPages();

app.Run();

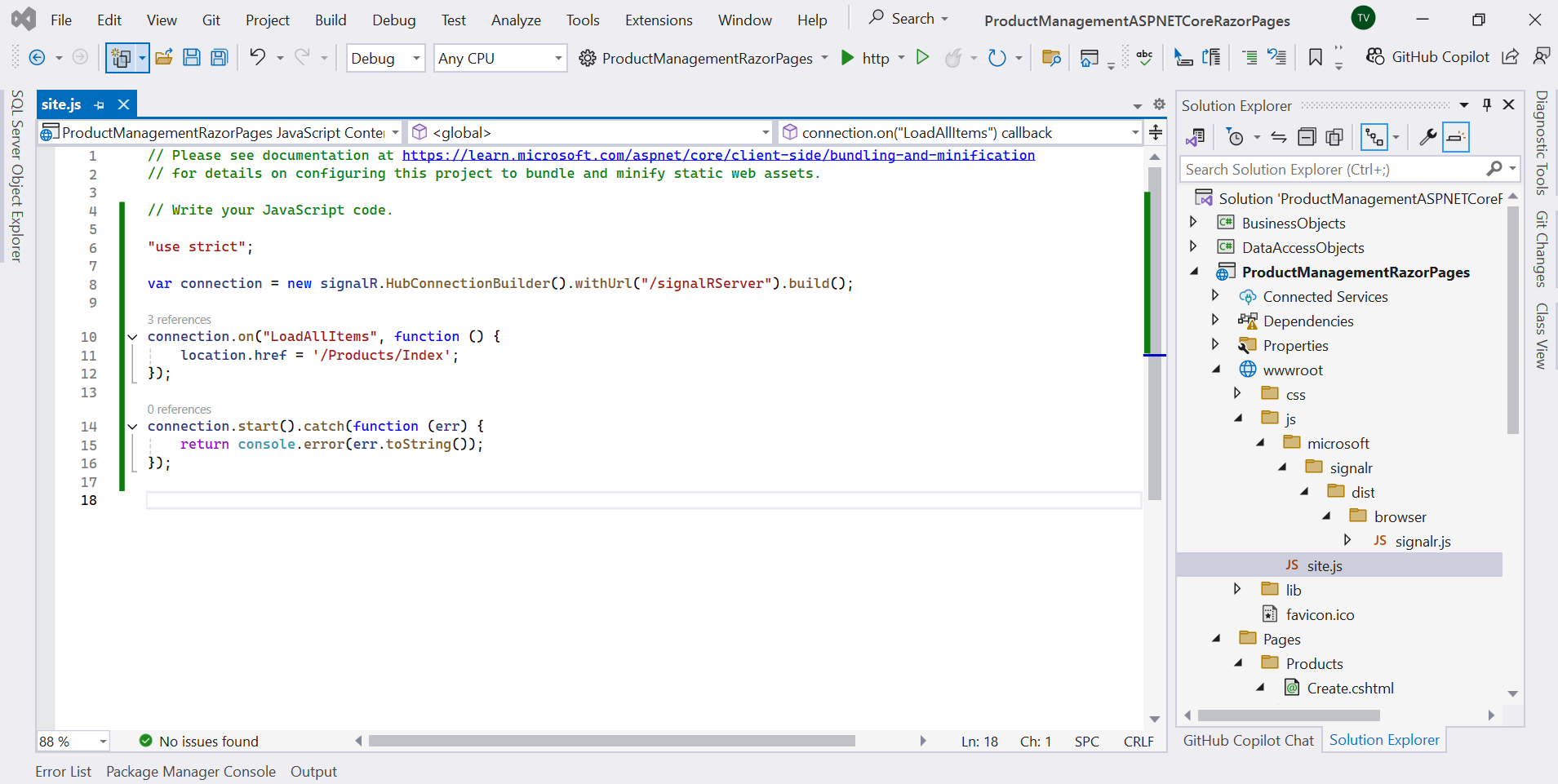
## **Step 03.** Add the SignalR client library

*The SignalR server library is included in the ASP.NET Core shared framework. The JavaScript client library isn't automatically included in the project. Use Library Manager (LibMan) to get the client library from unpkg.* ***unpkg*** *is a fast, global content delivery network for everything on* ***npm***





## **Step 04.** Add SignalR client code



***site.js***

"use strict";

var connection = new signalR.HubConnectionBuilder().withUrl("/signalRServer").build();

connection.on("LoadAllItems", function () {

location.href = '/Products/Index';

});

connection.start().catch(function (err) {

return console.error(err.toString());

});

## **Step 05.** Create a new product(/Pages/Products/Create.cshtml.cs), automatically update the list of product (/Pages/Products/Index.cshtml)

***/Pages/Products/Create.cshtml.cs***

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.RazorPages;

using Microsoft.AspNetCore.Mvc.Rendering;

using BusinessObjects;

using Services;

using Microsoft.AspNetCore.SignalR;

namespace ProductManagementRazorPages.Pages.Products

{

public class CreateModel : PageModel

{

private readonly IProductService \_contextProduct;

private readonly ICategoryService \_contextCategory;

private readonly IHubContext<SignalRServer> \_hubContext;

public CreateModel(IProductService context, ICategoryService categoryService, IHubContext<SignalRServer> hubContext)

{

\_contextProduct = context;

\_contextCategory = categoryService;

\_hubContext = hubContext;

}

public IActionResult OnGet()

{

ViewData["CategoryId"] = new SelectList(\_contextCategory.GetCategories(), "CategoryId", "CategoryId");

return Page();

}

[BindProperty]

public Product Product { get; set; } = default!;

// For more information, see https://aka.ms/RazorPagesCRUD.

public async Task<IActionResult> OnPostAsync()

{

if (!ModelState.IsValid)

{

return Page();

}

\_contextProduct.SaveProduct(Product);

await \_hubContext.Clients.All.SendAsync("LoadAllItems");

return RedirectToPage("./Index");

}

}

}

***/Pages/Products/Index.cshtml***

@page

@model ProductManagementRazorPages.Pages.Products.IndexModel

@{

ViewData["Title"] = "Index";

}

<h1>Index</h1>

<p>

<**a** **asp-page**="Create">Create New</**a**>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Product[0].ProductName)

</th>

<th>

@Html.DisplayNameFor(model => model.Product[0].UnitsInStock)

</th>

<th>

@Html.DisplayNameFor(model => model.Product[0].UnitPrice)

</th>

<th>

@Html.DisplayNameFor(model => model.Product[0].Category)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model.Product) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.ProductName)

</td>

<td>

@Html.DisplayFor(modelItem => item.UnitsInStock)

</td>

<td>

@Html.DisplayFor(modelItem => item.UnitPrice)

</td>

<td>

@Html.DisplayFor(modelItem => item.Category.CategoryName)

</td>

<td>

<**a** **asp-page**="./Edit" **asp-route-id**="@item.ProductId">Edit</**a**> |

<**a** **asp-page**="./Details" **asp-route-id**="@item.ProductId">Details</**a**> |

<**a** **asp-page**="./Delete" **asp-route-id**="@item.ProductId">Delete</**a**>

</td>

</tr>

}

</tbody>

</table>

<**script** src="~/js/microsoft/signalr/dist/browser/signalr.js"></**script**>

<**script** src="~/js/site.js"></**script**>

