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)

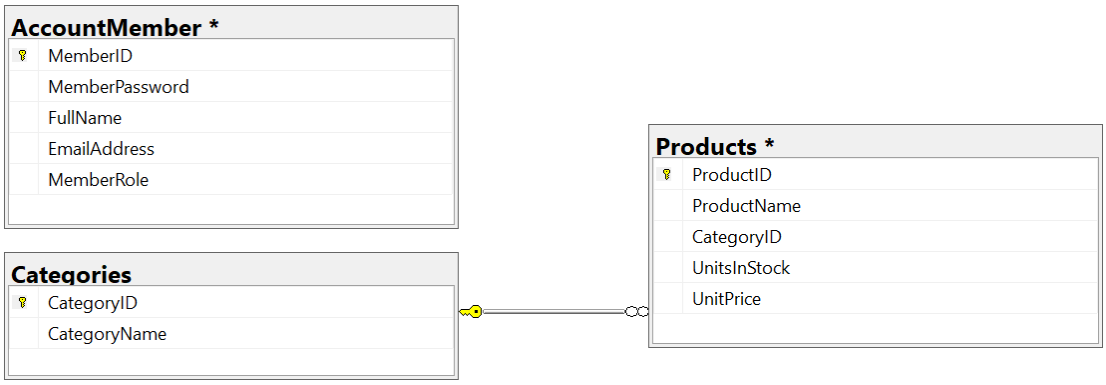


Table AccountMember

TV\SQLEXPRESS01...bo.AccountMember*

Column Name	Data Type	Allow Nulls
MemberID	nvarchar(20)	☐
MemberPassword	nvarchar(80)	☐
FullName	nvarchar(80)	☐
EmailAddress	nvarchar(100)	☐
MemberRole	int	☐
		☐

Table Categories

TV\SQLEXPRESS01.M...- dbo.Categories

Column Name	Data Type	Allow Nulls
CategoryID	int	☐
CategoryName	nvarchar(15)	☐
		☐

Table Products

TV\SQLEXPRESS01.M...re - dbo.Products*

Column Name	Data Type	Allow Nulls
ProductID	int	☐
ProductName	nvarchar(40)	☐
CategoryID	int	☐
UnitsInStock	smallint	☑
UnitPrice	money	☑
		☐

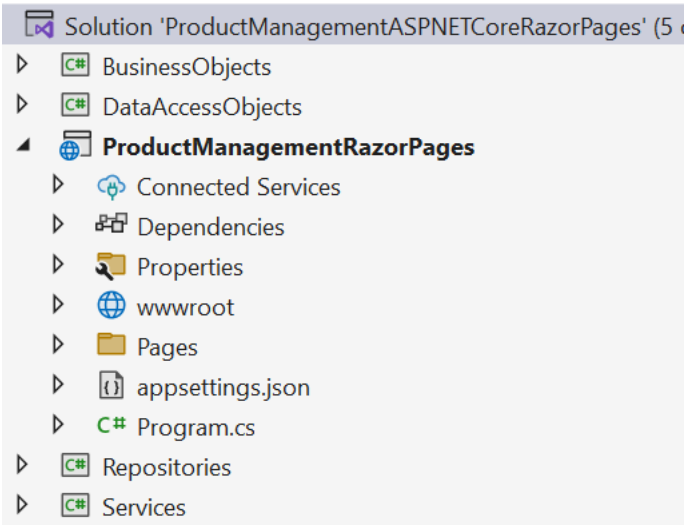
Activity 01: Build a solution by Visual Studio.NET

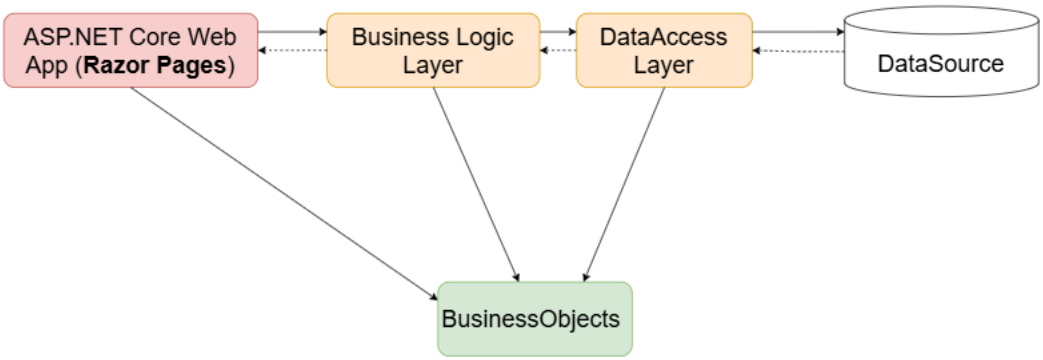
Create a Blank Solution named **ProductManagementASPNETCoreRazorPages** then add new a **Class Library** project named **BusinessObjects, DataAccessObjects, Repositories, Services** and an ASP.NET Core Web App (Razor Pages) project named **ProductManagementRazorPages**

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:

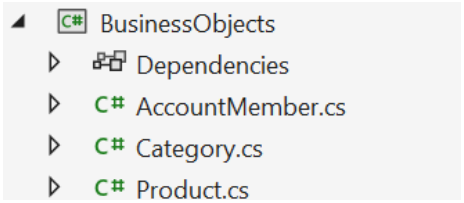
- Implement ORM

  dotnet ef dbcontext scaffold "**Server=(local); Database=MyStore; Uid=sa; Pwd=1234567890**" Microsoft.EntityFrameworkCore.SqlServer --output-dir ./

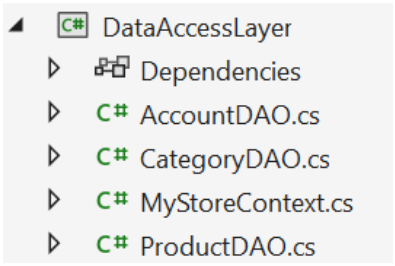
- Change the connection string in OnConfiguring() function of MyStoreContext.cs

```
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:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using BusinessObjects;

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

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

```
1   using BusinessObjects;
2   using Microsoft.EntityFrameworkCore;
3
4   namespace DataAccessObjects
5   {
        5 references
6       public class ProductDAO
7       {
            1 reference
8           public static List<Product> GetProducts()...
            1 reference
20          public static void SaveProduct(Product p)...
34
            1 reference
35          public static void UpdateProduct(Product p)...
48
            1 reference
49          public static void DeleteProduct(Product p)...
64
            1 reference
65          public static Product GetProductById(int id)...
71      }
72  }
```

The details of functions in ProductDAO.cs:

```
11      public static List<Product> GetProducts()
12      {
13          var listProducts = new List<Product>();
14          try
15          {
16              using var db = new MyStoreContext();
17              listProducts = db.Products.Include(f=>f.Category).ToList();
18              //using Microsoft.EntityFrameworkCore in order to use Include()
19          }
20          catch (Exception e) { }
21          return listProducts;
22      }

24      public static void SaveProduct(Product p)
25      {
26          try
27          {
28              using var context = new MyStoreContext();
29              context.Products.Add(p); // Add to Product collection
30              context.SaveChanges(); // Update Database
31          }
32          catch (Exception e)
33          {
34
35              throw new Exception(e.Message);
36          }
37      }

39      public static void UpdateProduct(Product p)
40      {
41          try
42          {
43              using var context = new MyStoreContext();
44              context.Entry<Product>(p).State
45                  = Microsoft.EntityFrameworkCore.EntityState.Modified;
46              context.SaveChanges();
47          }
48          catch (Exception e)
49          {
50              throw new Exception(e.Message);
51          }
52      }
```

```
54          public static void DeleteProduct(Product p)
55          {
56              try
57              {
58                  using var context = new MyStoreContext();
59                  var p1 =
60                      context.Products.SingleOrDefault(c => c.ProductId == p.ProductId);
61                  context.Products.Remove(p1);
62
63                  context.SaveChanges();
64              }
65              catch (Exception e)
66              {
67                  throw new Exception(e.Message);
68              }
69          }
```

```
71          public static Product GetProductById(int id)
72          {
73              using var db = new MyStoreContext();
74              return db.Products.FirstOrDefault(c => c.ProductId.Equals(id));
75
76          }
77      }
```

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

```
1   using BusinessObjects;
2
3   namespace DataAccessObjects
4   {
        1 reference
5       public class AccountDAO
6       {
            1 reference
7           public static AccountMember GetAccountById(string accountID)
8           {
9               using var db = new MyStoreContext();
10              return db.AccountMembers.FirstOrDefault(c=>c.MemberId.Equals(accountID)) ;
11          }
12      }
13  }
```

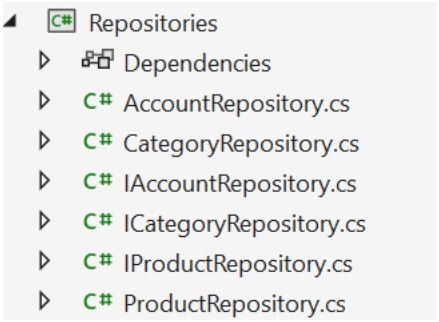
Step 04. The codes for **MyStoreContext.cs**:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using BusinessObjects;

#nullable disable

namespace DataAccessObjects
{
    10 references
    public partial class MyStoreContext : DbContext
    {
        7 references
        public MyStoreContext()...

        0 references
        public MyStoreContext(DbContextOptions<MyStoreContext> options)...

        1 reference
        public virtual DbSet<AccountMember> AccountMembers { get; set; } = null!;
        1 reference
        public virtual DbSet<Category> Categories { get; set; } = null!;
        5 references
        public virtual DbSet<Product> Products { get; set; } = null!;

        0 references
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)...

        1 reference
        string GetConnectionString()...

        0 references
        protected override void OnModelCreating(ModelBuilder modelBuilder)...

        1 reference
        partial void OnModelCreatingPartial(ModelBuilder modelBuilder);
    }
}
```

The details for GetConnectionString() and OnConfiguring() functions

```
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (!optionsBuilder.IsConfigured)
            {
#warning To protect potentially sensitive information in your connection string, you s
                optionsBuilder.UseSqlServer(GetConnectionString());
            }
        }
        1 reference
        string GetConnectionString()
        {
            IConfiguration config = new ConfigurationBuilder()
                .SetBasePath(Directory.GetCurrentDirectory())
                .AddJsonFile("appsettings.json").Build();
            return config["ConnectionStrings:MyStockDB"];

        }
```

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:

```
1   using System.Collections.Generic;
2   using BusinessObjects;
3
4   namespace Repositories
5   {
        2 references
6       public interface ICategoryRepository
7       {
            2 references
8           List<Category> GetCategories();
9       }
10  }
```

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

```
using System.Collections.Generic;
using BusinessObjects;

namespace Repositories
{
    2 references
    public interface IProductRepository
    {
        2 references
        void SaveProduct(Product p);
        2 references
        void DeleteProduct(Product p);
        2 references
        void UpdateProduct(Product p);
        2 references
        List<Product> GetProducts();
        2 references
        Product GetProductById(int id);
    }
}
```

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

```
using BusinessObjects;

namespace Repositories
{
    2 references
    public interface IAccountRepository
    {
        2 references
        AccountMember GetAccountById(string accountID);
    }
}
```

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

```
using BusinessObjects;
using DataAccessObjects;

namespace Repositories
{
    1 reference
    public class CategoryRepository : ICategoryRepository
    {
        2 references
        public List<Category> GetCategories() => CategoryDAO.GetCategories();
    }
}
```

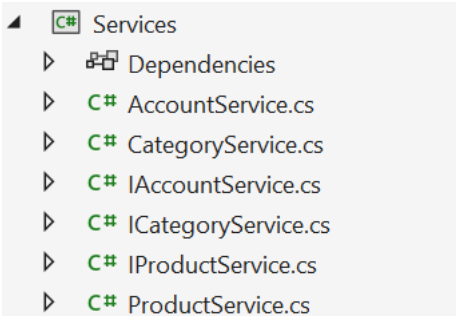
Step 05. Write codes for class **ProductRepository.cs** as follows:

```
using BusinessObjects;
using DataAccessObjects;

namespace Repositories
{
    1 reference
    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);
        2 references
        public List<Product> GetProducts() => ProductDAO.GetProducts();
        2 references
        public Product GetProductById(int id)=>ProductDAO.GetProductById(id);

    }
}
```

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

```
using BusinessObjects;
using DataAccessObjects;

namespace Repositories
{
    1 reference
    public class AccountRepository : IAccountRepository
    {
        2 references
        public AccountMember GetAccountById(string accountID)
            =>AccountDAO.GetAccountById(accountID);

    }
}
```

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:

```
using BusinessObjects;
using System.Collections.Generic;
namespace Services
{
    2 references
    public interface ICategoryService
    {
        2 references
        List<Category> GetCategories();
    }
}
```

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

```
using BusinessObjects;
using System.Collections.Generic;
namespace Services
{
    2 references
    public interface IProductService
    {
        2 references
        void SaveProduct(Product p);
        2 references
        void DeleteProduct(Product p);
        2 references
        void UpdateProduct(Product p);
        2 references
        List<Product> GetProducts();
        2 references
        Product GetProductById(int id);
    }
}
```

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

```
using BusinessObjects;

namespace Services
{
    2 references
    public interface IAccountService
    {
        2 references
        AccountMember GetAccountById(string accountID);
    }
}
```

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

```
using System.Collections.Generic;
namespace Services
{
    2 references
    public class CategoryService : ICategoryService
    {
        private readonly ICategoryRepository iCategoryRepository;

        1 reference
        public CategoryService()
        {
            iCategoryRepository = new CategoryRepository();
        }

        2 references
        public List<Category> GetCategories()
        {
            return iCategoryRepository.GetCategories();
        }
    }
}
```

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

```
using BusinessObjects;
using Repositories;
using System.Collections.Generic;

namespace Services
{
    2 references
    public class ProductService : IProductService
    {
        private readonly IProductRepository iProductRepository;

        1 reference
        public ProductService()
        {
            iProductRepository = new ProductRepository();
        }
        2 references
        public void DeleteProduct(Product p)
        {
            iProductRepository.DeleteProduct(p);
        }

        2 references
        public Product GetProductById(int id)
        {
            return iProductRepository.GetProductById(id);
        }

        2 references
        public List<Product> GetProducts()
        {
            return iProductRepository.GetProducts();
        }

        2 references
        public void SaveProduct(Product p)
        {
            iProductRepository.SaveProduct(p);
        }

        2 references
        public void UpdateProduct(Product p)
        {
            iProductRepository.UpdateProduct(p);
        }
    }
}
```

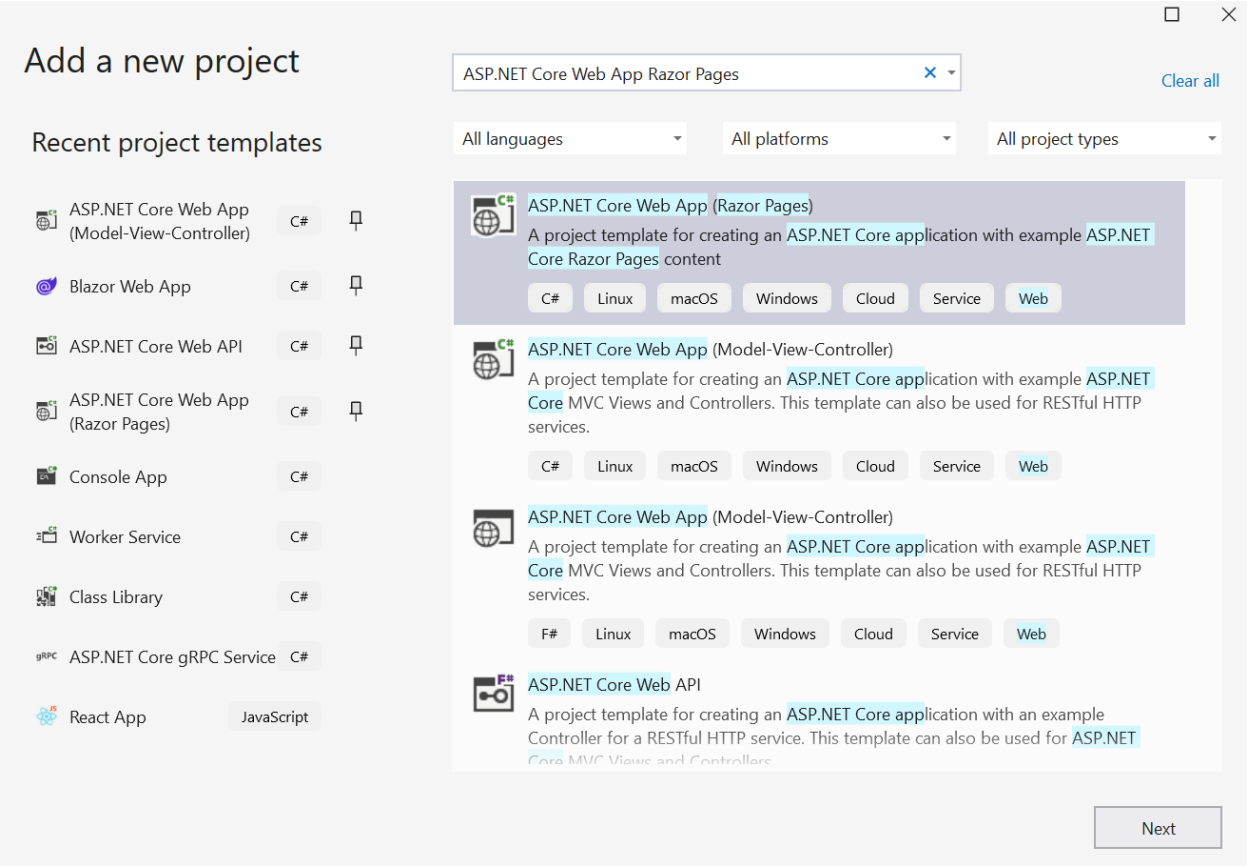
Step 06. Write codes for class **AccountService.cs** as follows:

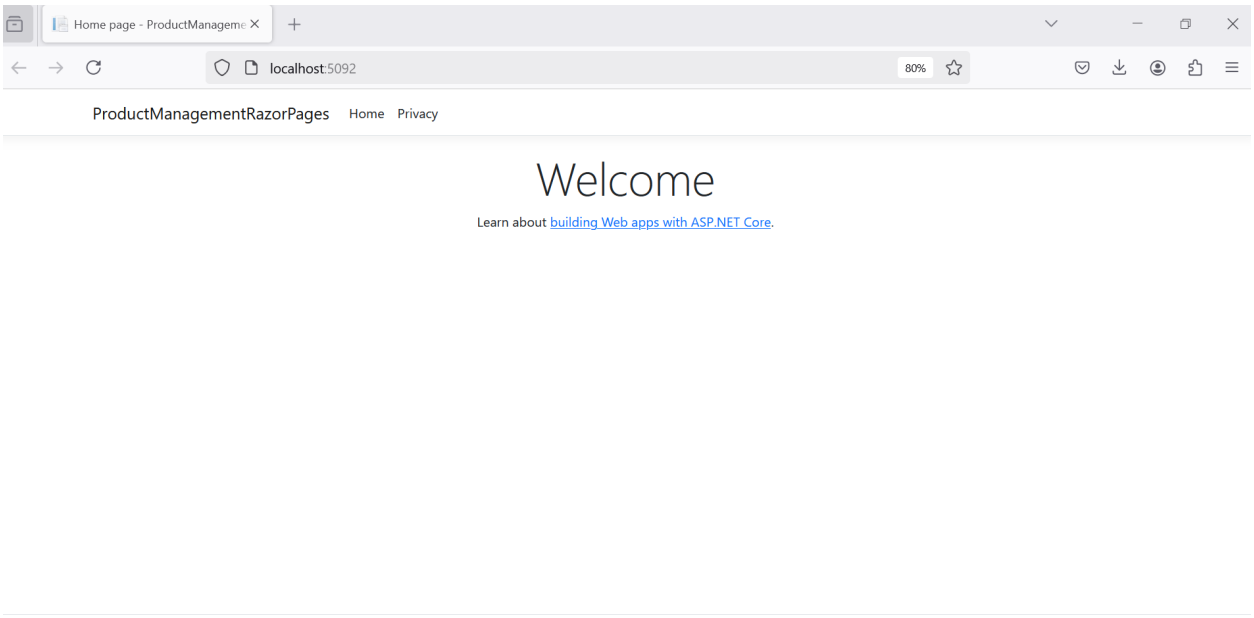
```
1   using BusinessObjects;
2   using Repositories;
3
4   namespace Services
5   {
        2 references
6       public class AccountService : IAccountService
7       {
8           private readonly IAccountRepository iAccountRepository;
            1 reference
9           public AccountService() {
10              iAccountRepository = new AccountRepository();
11          }
            2 references
12          public AccountMember GetAccountById(string accountID)
13          {
14              return iAccountRepository.GetAccountById(accountID);
15          }
16      }
17  }
```

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)




Home page - ProductManageme
localhost:5092
80%
ProductManagementRazorPages
Home
Privacy
Welcome
Learn about building Web apps with ASP.NET Core.
© 2025 - ProductManagementRazorPages - Privacy

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

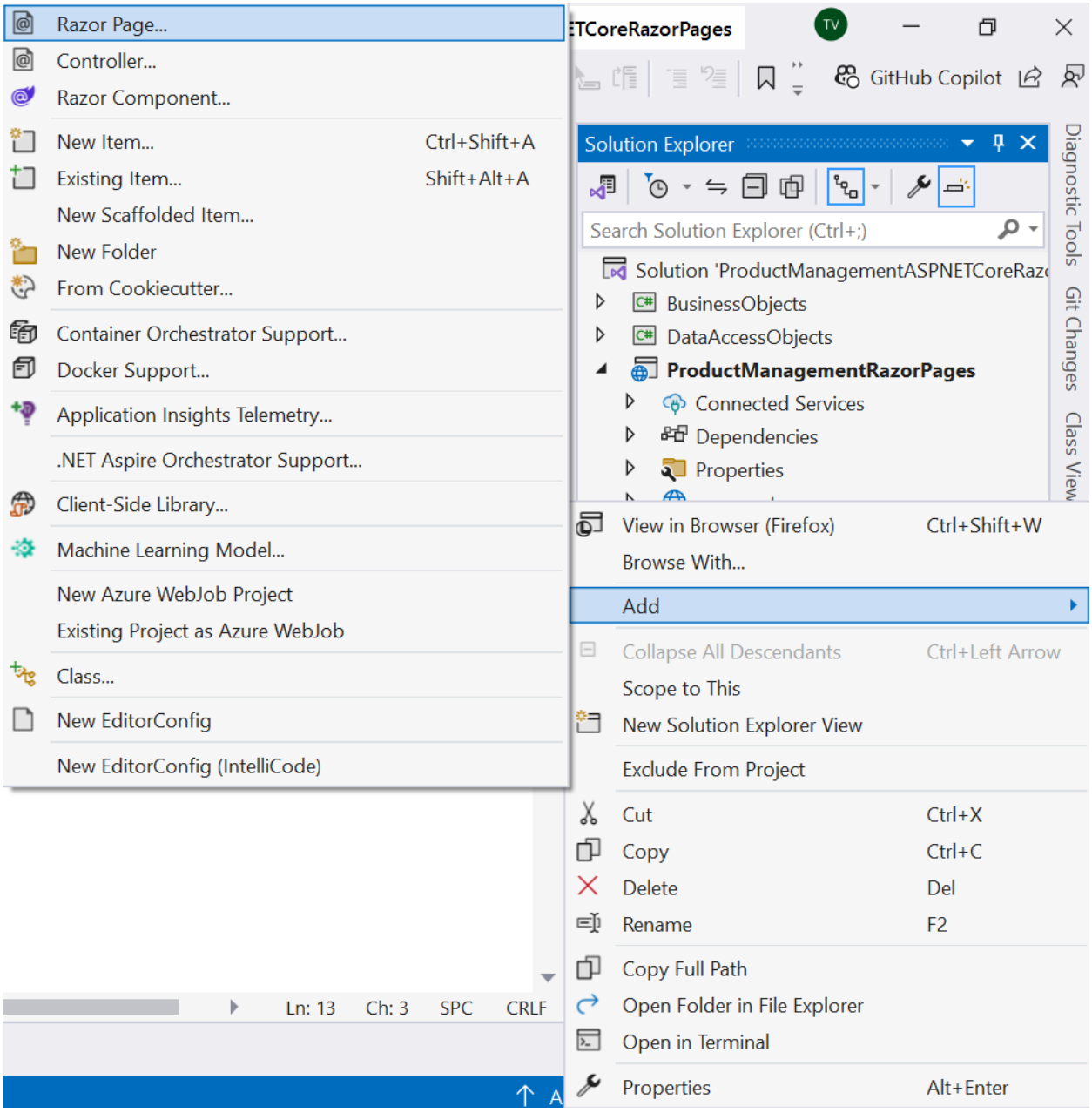
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MyStockDB":
"Server=localhost;uid=sa;pwd=1234567890;database=MyStore;TrustServerCertificate=True"
  }
}
```

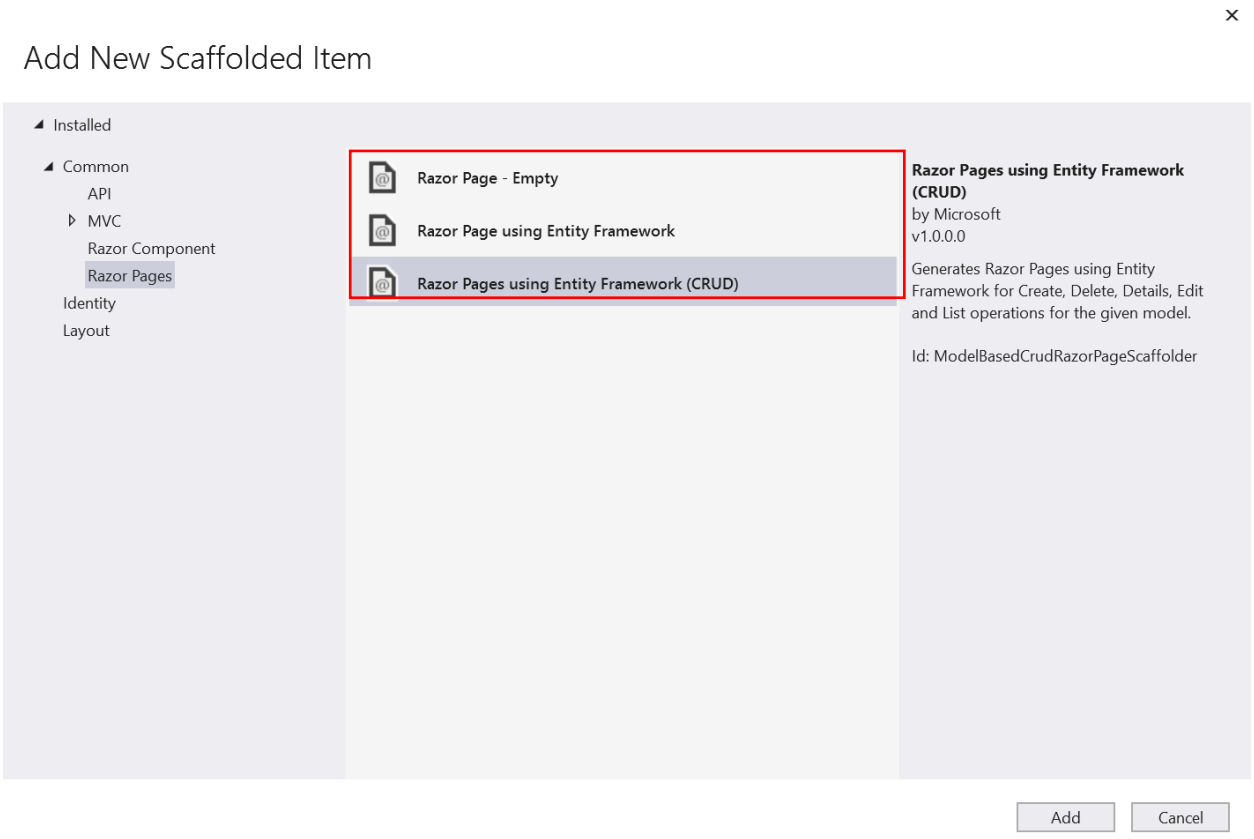
Step 03. Add **Business Objects** and **Services** projects as references for the ASP.NET Core Web App (Razor Pages)

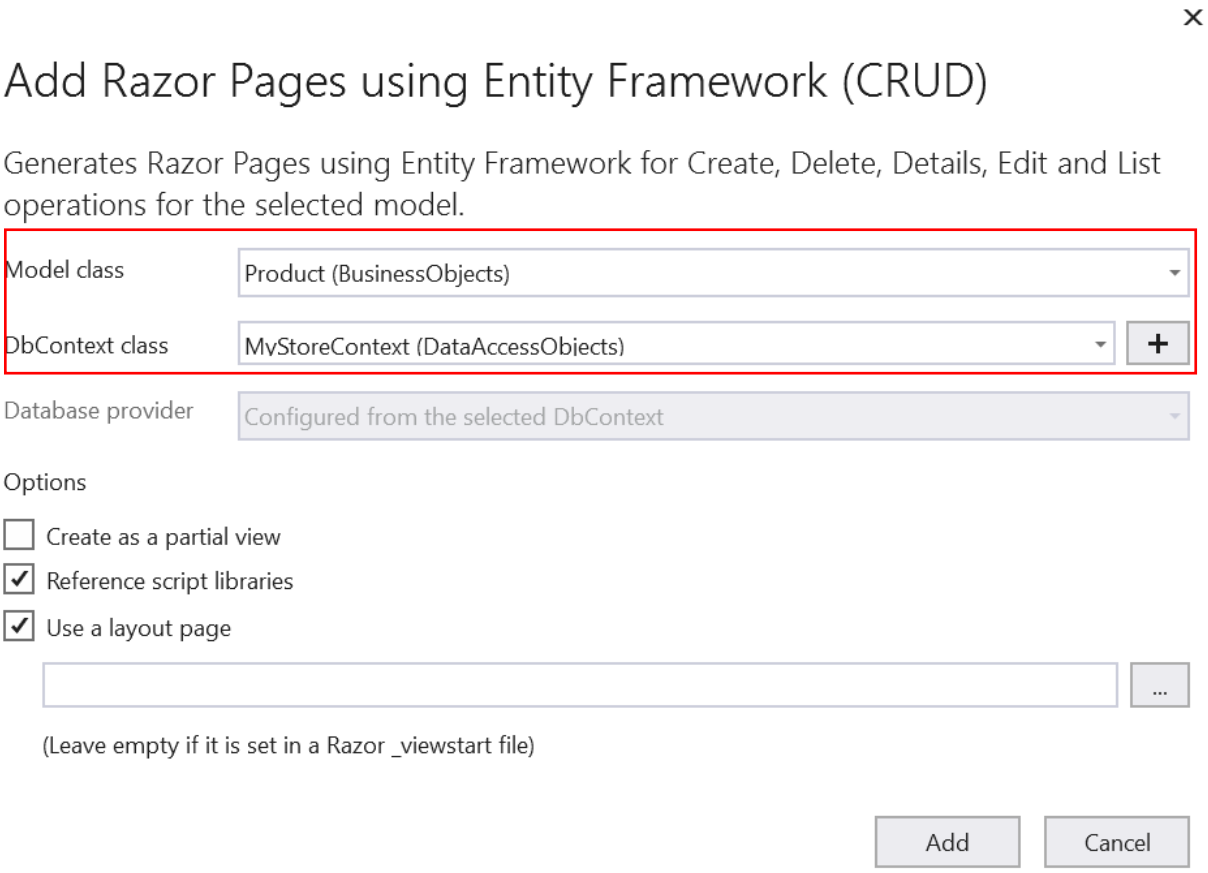
Step 04. Create Razor Pages

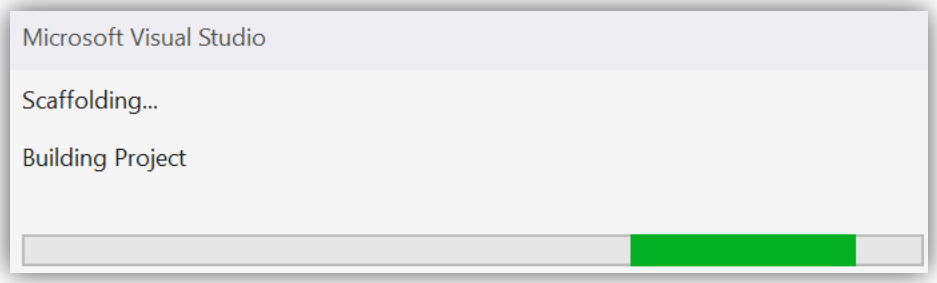
- Connect direct to the Data Access Layer (MyStoreContext.cs) to generate code
- Then
  - Add Dependency Injection
  - Change the code connects to the Service Layer

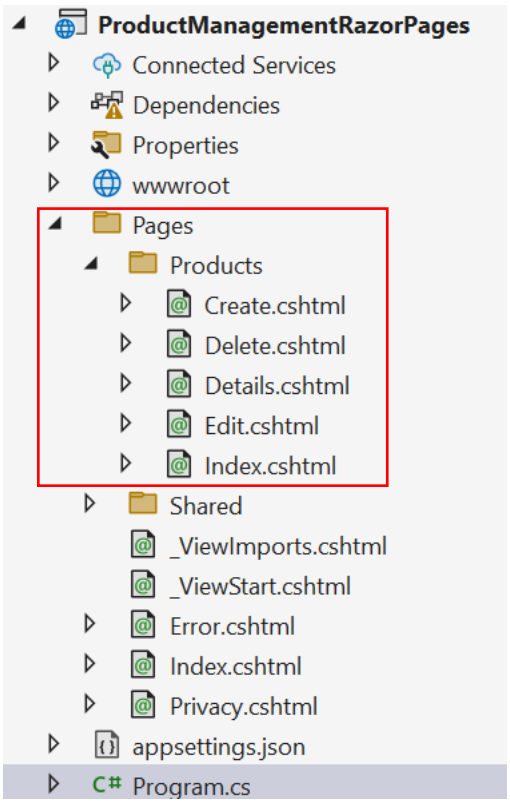
Add new Razor Pages




Add New Scaffolded Item
Installed
Common
API
MVC
Razor Component
Razor Pages
Identity
Layout
Razor Page - Empty
Razor Page using Entity Framework
Razor Pages using Entity Framework (CRUD)
Razor Pages using Entity Framework (CRUD)
by Microsoft
v1.0.0.0
Generates Razor Pages using Entity Framework for Create, Delete, Details, Edit and List operations for the given model.
Id: ModelBasedCrudRazorPageScaffolder
Add
Cancel

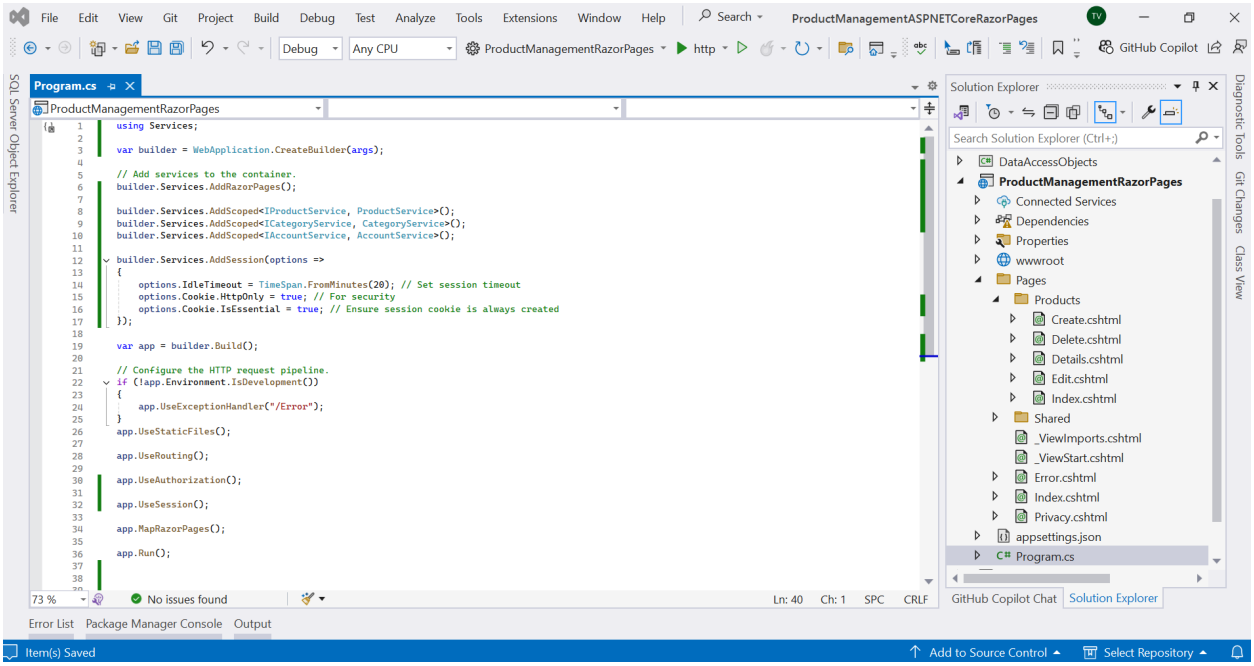

Add Razor Pages using Entity Framework (CRUD)
Generates Razor Pages using Entity Framework for Create, Delete, Details, Edit and List operations for the selected model.
Model class
Product (BusinessObjects)
DbContext class
MyStoreContext (DataAccessObjects)
Database provider
Configured from the selected DbContext
Options
Create as a partial view
Reference script libraries
Use a layout page
(Leave empty if it is set in a Razor _viewstart file)
Add
Cancel


Microsoft Visual Studio
Scaffolding...
Building Project



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;
```

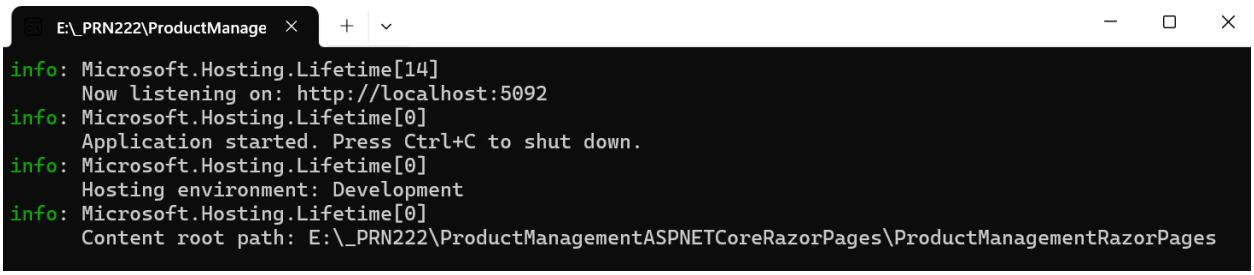
```
namespace ProductManagementRazorPages.Pages.Products
{
    public class DetailsModel : PageModel
    {
        private readonly IProductService _context;

        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();
```

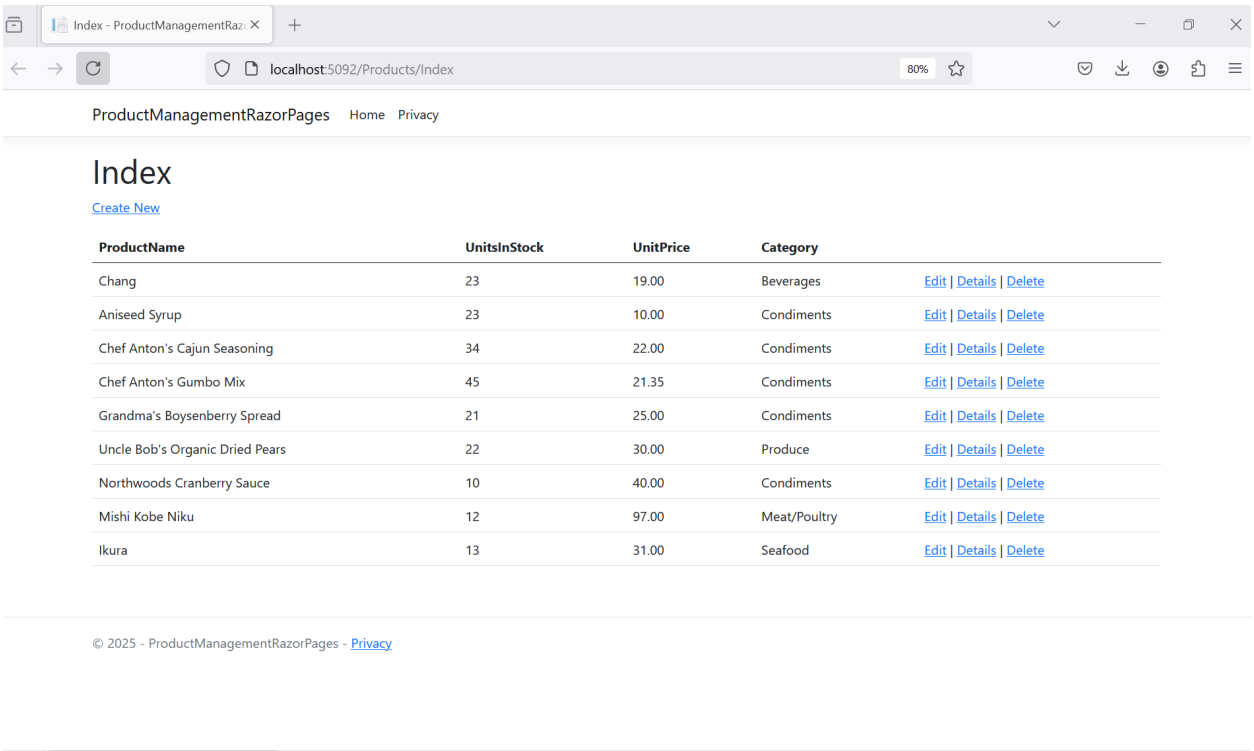
```
    }
  }
}
```

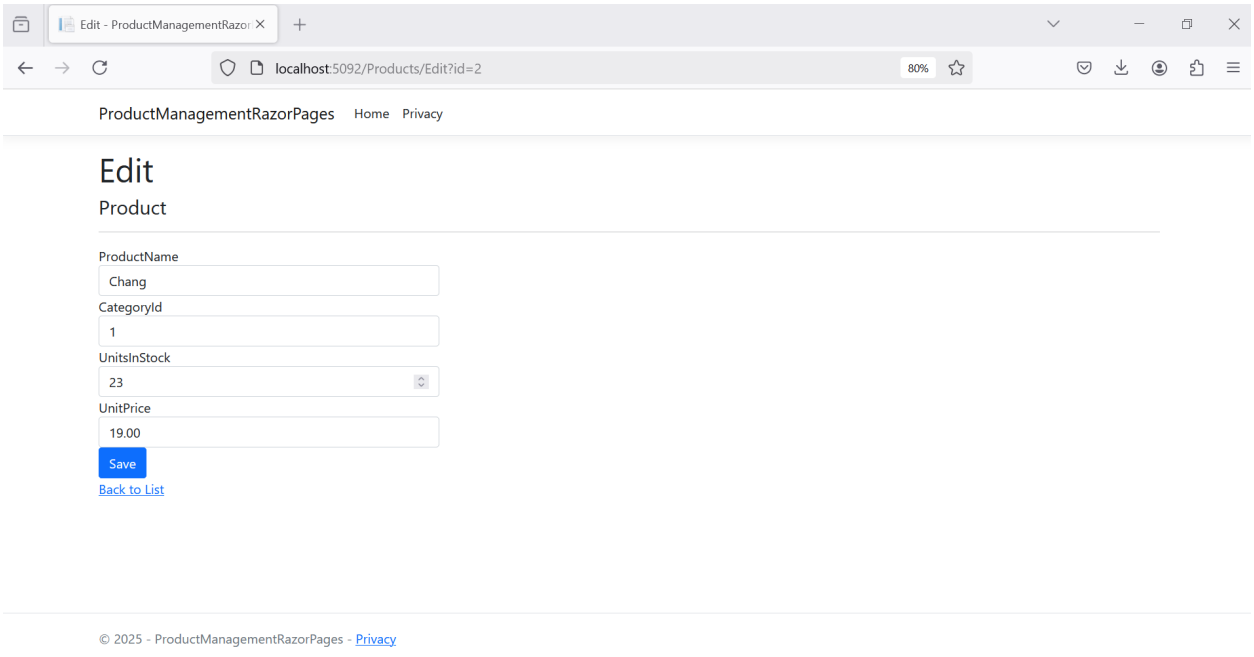
Step 06. Change the View in the Razor View depend on your template.

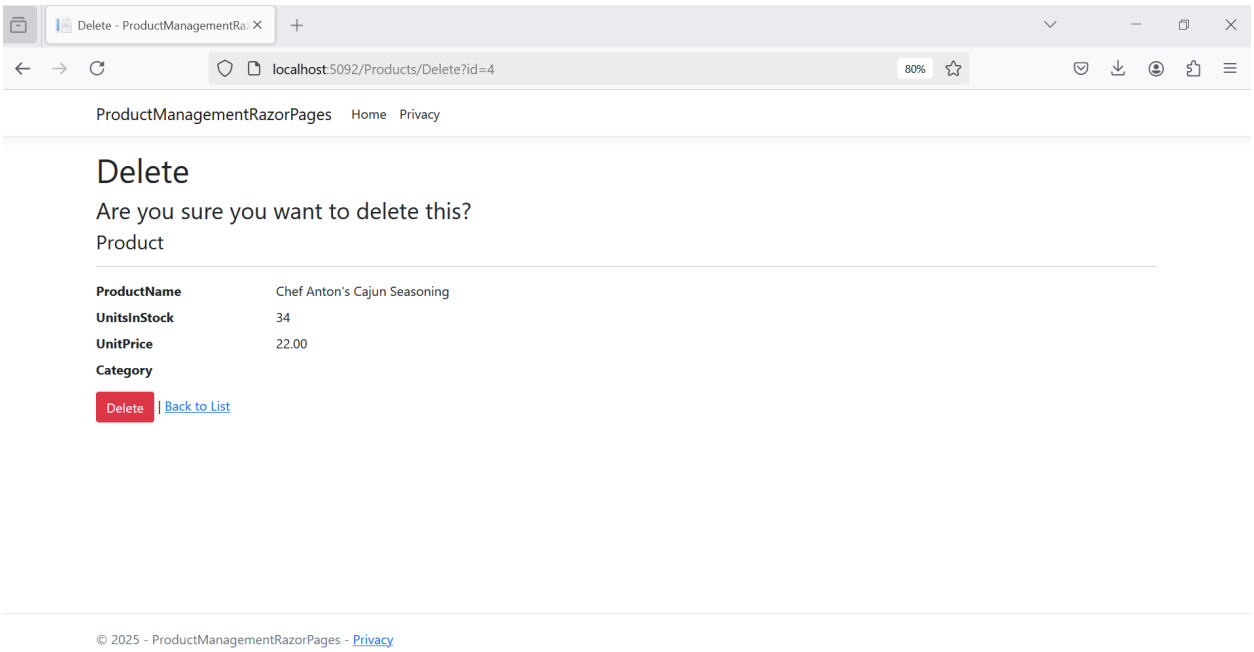
Step 07. Clean ASP.NET Core Web App (Razor Pages) project, remove Entity Framework Core related packages.

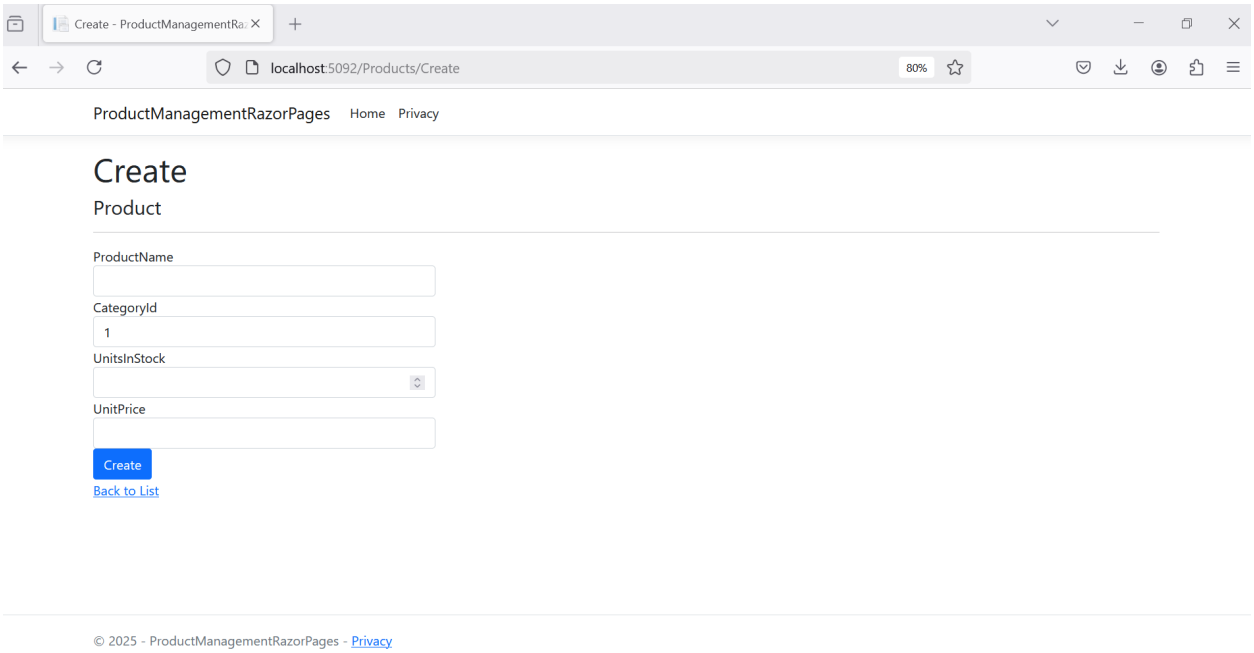
Activity 07: Run the ASP.NET Core Web App (Razor Pages) project and test all actions




Index - ProductManagementRaz
localhost:5092/Products/Index
80%
ProductManagementRazorPages
Home
Privacy
Index
Create New
ProductName
UnitsInStock
UnitPrice
Category
Chang 23 19.00 Beverages Edit | Details | Delete
Aniseed Syrup 23 10.00 Condiments Edit | Details | Delete
Chef Anton's Cajun Seasoning 34 22.00 Condiments Edit | Details | Delete
Chef Anton's Gumbo Mix 45 21.35 Condiments Edit | Details | Delete
Grandma's Boysenberry Spread 21 25.00 Condiments Edit | Details | Delete
Uncle Bob's Organic Dried Pears 22 30.00 Produce Edit | Details | Delete
Northwoods Cranberry Sauce 10 40.00 Condiments Edit | Details | Delete
Mishi Kobe Niku 12 97.00 Meat/Poultry Edit | Details | Delete
Ikura 13 31.00 Seafood Edit | Details | Delete
© 2025 - ProductManagementRazorPages - Privacy


Edit - ProductManagementRazor
localhost:5092/Products/Edit?id=2
80%
ProductManagementRazorPages
Home
Privacy
Edit
Product
ProductName
Chang
CategoryId
1
UnitsInStock
23
UnitPrice
19.00
Save
Back to List
© 2025 - ProductManagementRazorPages - Privacy


Delete - ProductManagementRa
localhost:5092/Products/Delete?id=4
80%
ProductManagementRazorPages
Home
Privacy
Delete
Are you sure you want to delete this?
Product
ProductName
Chef Anton's Cajun Seasoning
UnitsInStock
34
UnitPrice
22.00
Category
Delete
| Back to List
© 2025 - ProductManagementRazorPages - Privacy


Create - ProductManagementRa
localhost:5092/Products/Create
80%
ProductManagementRazorPages
Home
Privacy
Create
Product
ProductName
CategoryId
1
UnitsInStock
UnitPrice
Create
Back to List
© 2025 - ProductManagementRazorPages - Privacy

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>
```

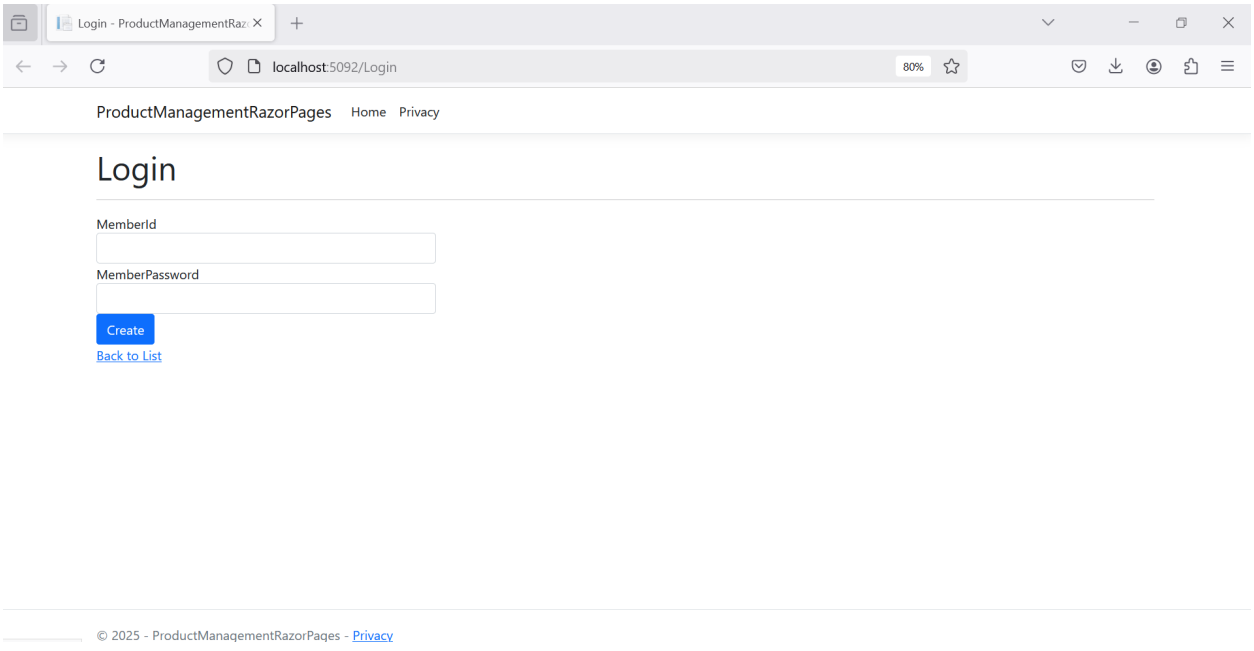
```
        <div class="form-group">
            <label asp-for="AccountMember.MemberId" class="control-label"></label>
            <input asp-for="AccountMember.MemberId" class="form-control" />
            <span asp-validation-for="AccountMember.MemberId" class="text-danger"></span>
        </div>
        <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");
```

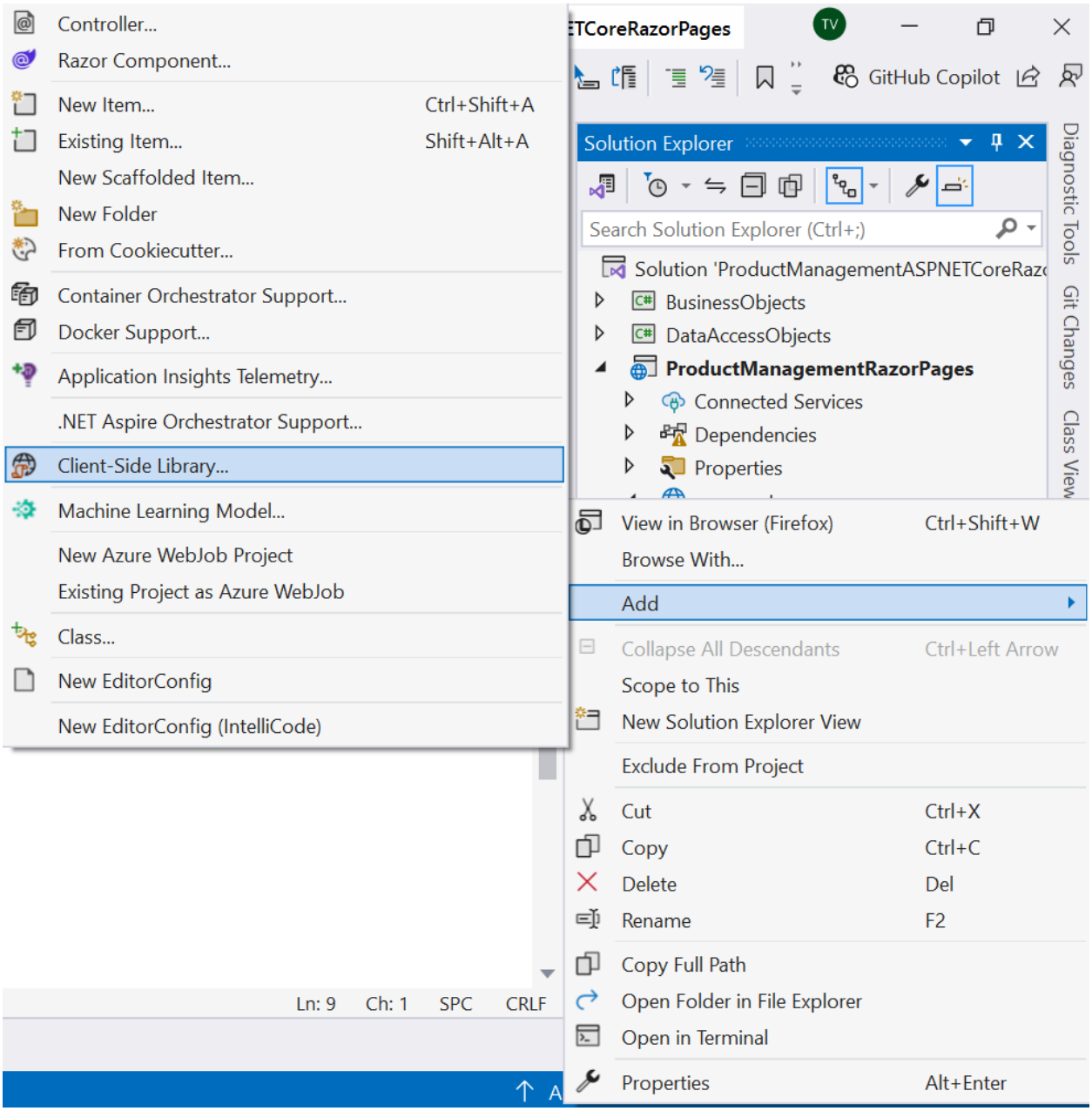
Program.cs

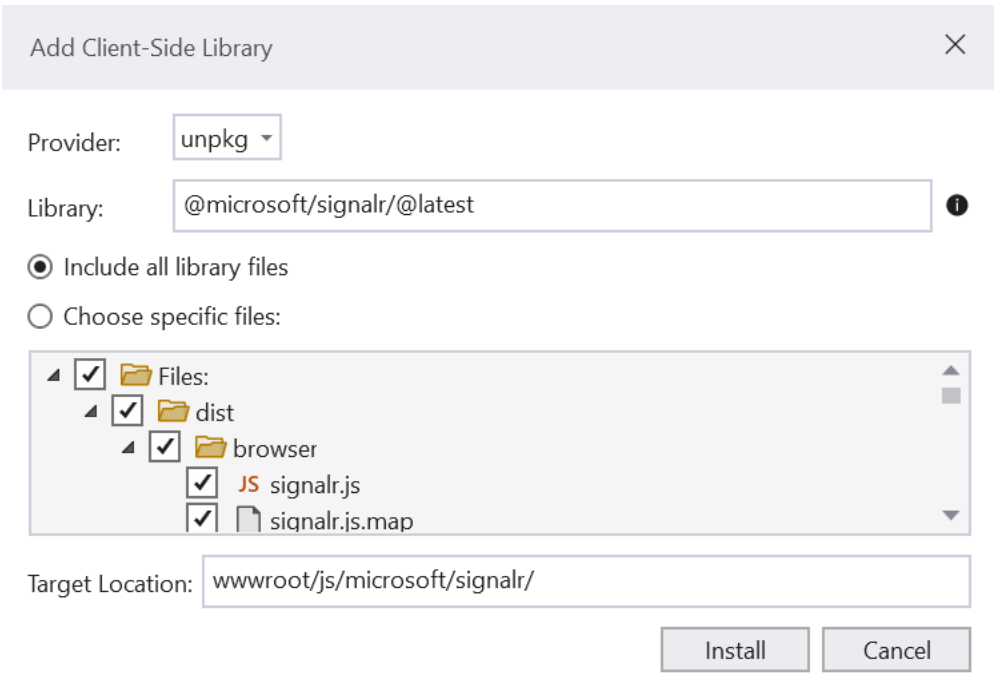
```
using ProductManagementRazorPages;
using Services;

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();
```

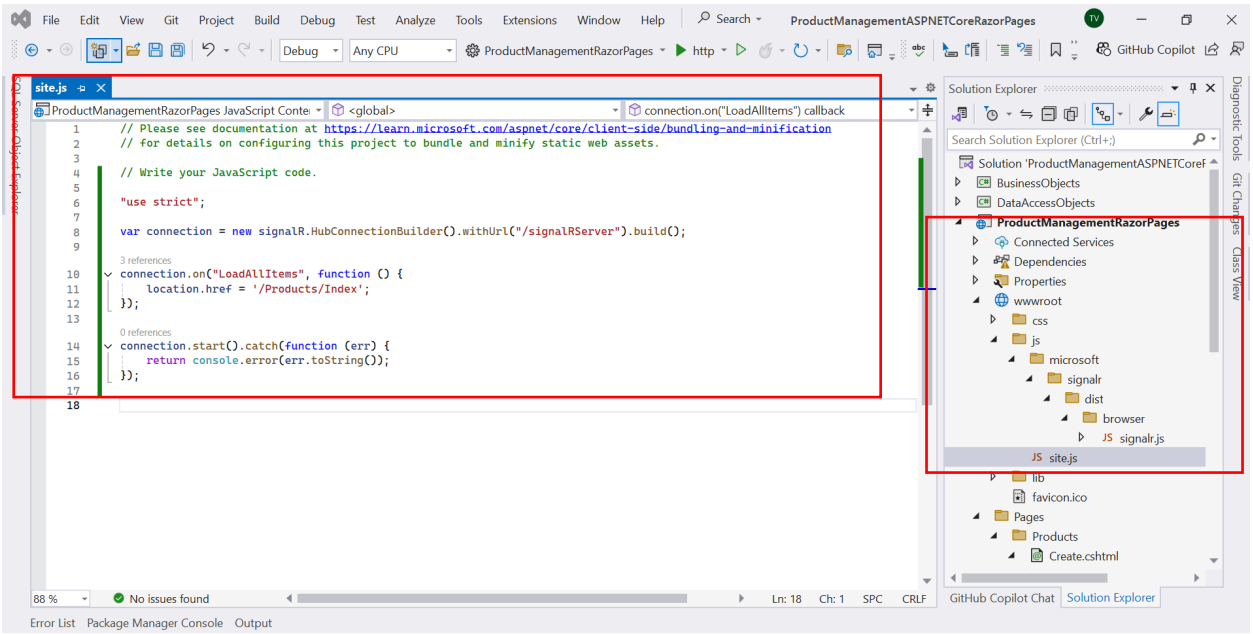
Step 03. Add the SignalR client library

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>
```