Cao Đẳng SÀI GÒN

SAIGONTECH
SAIGON INSTITUTE OF TECHNOLOGY

SportsStore – A real application

Bài 03: Part 1

Nội dung

- Tạo SportsStore ASP.NET CORE project
- Thêm data model và hỗ trợ cơ sở dữ liệu
- Hiển thị danh mục sản phẩm cơ bản
- Phân trang dữ liệu
- Định dạng nội dung trang

1 Creating the projects

- Tạo SportsStore project

```
dotnet new globaljson --sdk-version 7.0.401 --output SportsSln/SportsStore
dotnet new mvc --no-https --output SportsSln/SportsStore --framework net7.0
dotnet new sln -o SportsSln

dotnet sln SportsSln add SportsSln/SportsStore
```

1.1 Creating the unit test project

- Tạo unit test project

```
dotnet new xunit -o SportsSln/SportsStore.Tests --framework net7.0
dotnet sln SportsSln add SportsSln/SportsStore.Tests
dotnet add SportsSln/SportsStore.Tests reference SportsSln/SportsStore
```

- Cài đặt **Moq** package để tạo mock objects

```
dotnet add SportsSln/SportsStore.Tests package Moq --version 4.18.4
```

1.2 Opening the project

- Visual Studio 2022
- Visual Studio Code

1.3 Configuring the HTTP port

SportsStore/Properties/launchSettings.json

```
{
  "iisSettings": {
    "windowsAuthentication": false,
    "anonymousAuthentication": true,
    "iisExpress": {
      "applicationUrl": "http://localhost:5000",
      "sslPort": 0
    }
  },
  "profiles": {
    "http": {
      "commandName": "Project",
      "dotnetRunMessages": true,
      "launchBrowser": true,
      "applicationUrl": "http://localhost:5000",
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    },
    "IIS Express": {
      "commandName": "IISExpress",
      "launchBrowser": true,
      "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
      }
    }
  }
}
```

1.4 The application project folders

Name	Description
Models	This folder will contain the data model and the classes that provide access to the data in the application's database.
Controllers	This folder will contain the controller classes that handle HTTP requests.
Views	This folder will contain all the Razor files, grouped into separate subfolders.
Views/Home	This folder will contain Razor files that are specific to the Home controller, which I create in the "Creating the Controller and View" section.
Views/Shared	This folder will contain Razor files that are common to all controllers.

1.5 Preparing the service and the request pipeline

- **`AddControllersWithViews`**: thiết lập các đối tượng được chia sẻ mà các ứng dụng yêu cầu bằng các sử dụng MVC Framework và Razor view engine
- **`UseStaticFiles`**: cho phép truy cập các statis file trong thư mục wwwroot

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllersWithViews();

var app = builder.Build();

//app.MapGet('/', () => 'Hello World!');

app.UseStaticFiles();
app.MapDefaultControllerRoute();

app.Run();
```

1.6 Configuring the Razor view engine

Views/_ViewImports.cshtml

```
@using SportsStore.Models
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Views/_ViewStart.cshtml

```
@{
    Layout = "_Layout";
}
```

Views/Shared/_Layout.cshtml

```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>SportsStore</title>
</head>
<body>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

1.7 Creating the controller and view

Controllers/HomeController.cs

```
using System.Diagnostics;
using Microsoft.AspNetCore.Mvc;
using SportsStore.Models;

namespace SportsStore.Controllers;

public class HomeController : Controller {
    public IActionResult Index() => View();
}
```

Views/Index.cshtml

```
<h4>Welcome to SportsStore</h4>
```

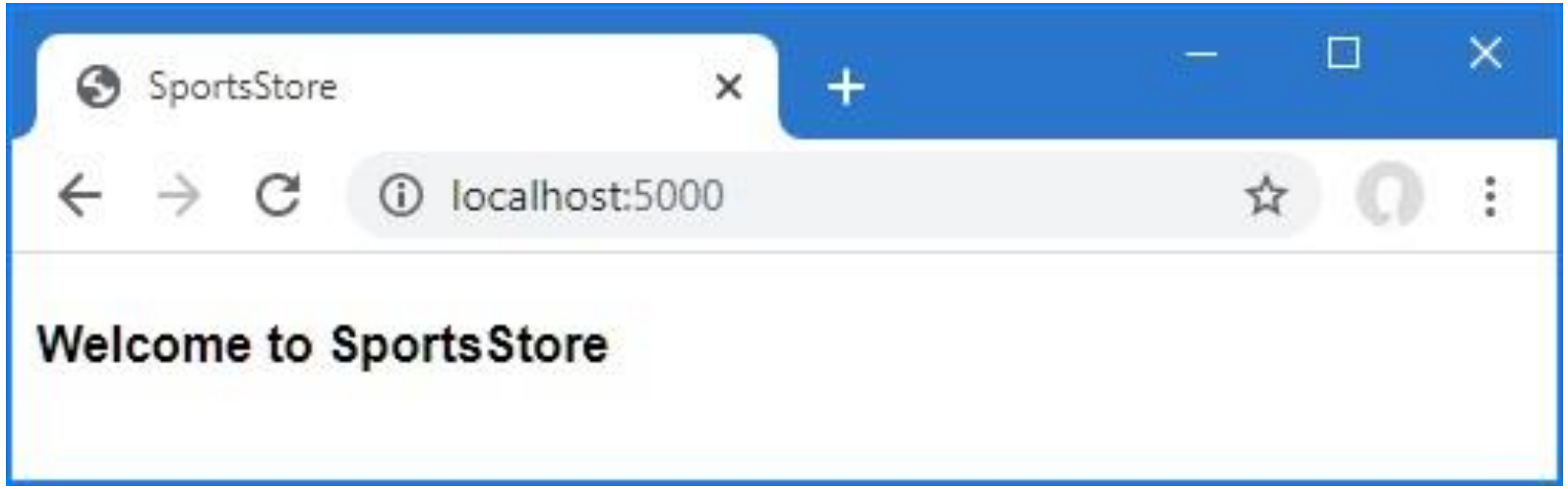
1.8 Starting the data model

Model/Product.cs

```
using System.ComponentModel.DataAnnotations.Schema;
namespace SportsStore.Models {
    public class Product {
        public long? ProductID { get; set; }

        public string Name { get; set; } = String.Empty;

        public string Description { get; set; } = String.Empty;

        [Column(TypeName = "decimal(8, 2)")]
        public decimal Price { get; set; }

        public string Category { get; set; } = String.Empty;
    }
}
```

1.9 Checking and running the application

- Power shell: dotnet run
- Request : http://localhost:5000



2 Adding data to the application

- Dữ liệu ứng dụng được lưu trong SQL Server LocalDB
- Lưu ý: Phải cài đặt LocalDB trong Visual Studio 2022

2.1 Installing the Entity Framework Core package

- Adding the Entity Framework Core package to the **SportsStore** project

```
dotnet add package Microsoft.EntityFrameworkCore.Design --version 7.0.13
dotnet add package Microsoft.EntityFrameworkCore.SqlServer --version 7.0.13
```

- Installing the Entity Framework Core tool package

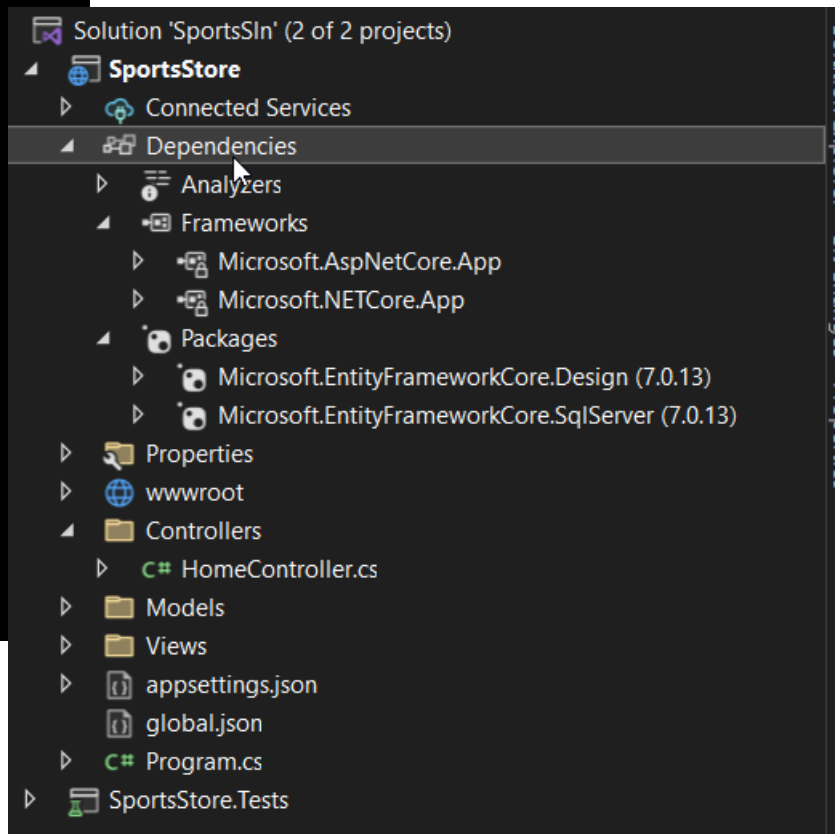
```
dotnet tool uninstall --global dotnet-ef
dotnet tool install --global dotnet-ef --version 7.0.13
```

2.1 Installing the Entity Framework Core package

- Thông tin Entity Framework và package trong SportsStore project

SportsStore.csproj

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.EntityFrameworkCore.Design" Version="7.0.13">
      <IncludeAssets>runtime; build; native; contentfiles; analyzers;
buildtransitive</IncludeAssets>
      <PrivateAssets>all</PrivateAssets>
    </PackageReference>
    <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.13" />
  </ItemGroup>

</Project>
```



2.2 Defining the connection string

appsettings.json

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "SportsStoreConnection":
"Server=(localdb)\\MSSQLLocalDB;Database=SportsStore;MultipleActiveResultSets=true"
  }
}
```

2.3 Creating the database context class

Models/StoreDbContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace SportsStore.Models {
    public class StoreDbContext : DbContext {
        public StoreDbContext(DbContextOptions<StoreDbContext> options)
            : base(options) { }

        public DbSet<Product> Products => Set<Product>();
    }
}
```

2.4 Configuring Entity Framework Core

Program.cs

```
using Microsoft.EntityFrameworkCore;
using SportsStore.Models;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<StoreDbContext>(opts => {
    opts.UseSqlServer(
        builder.Configuration["ConnectionStrings:SportsStoreConnection"]);
});

var app = builder.Build();
//app.MapGet('/', () => 'Hello World!');
app.UseStaticFiles();
app.MapDefaultControllerRoute();
app.Run();
```

2.5 Creating a repository

Models/IStoreRepository.cs

```
namespace SportsStore.Models {
    public interface IStoreRepository {

        IQueryable<Product> Products { get; }
    }
}
```

Models/EFStoreRepository.cs

```
namespace SportsStore.Models {
    public class EFStoreRepository : IStoreRepository {
        private StoreDbContext context;
        public EFStoreRepository(StoreDbContext ctx) {
            context = ctx;
        }
        public IQueryable<Product> Products => context.Products;
    }
}
```

2.5 Creating a repository

Program.cs

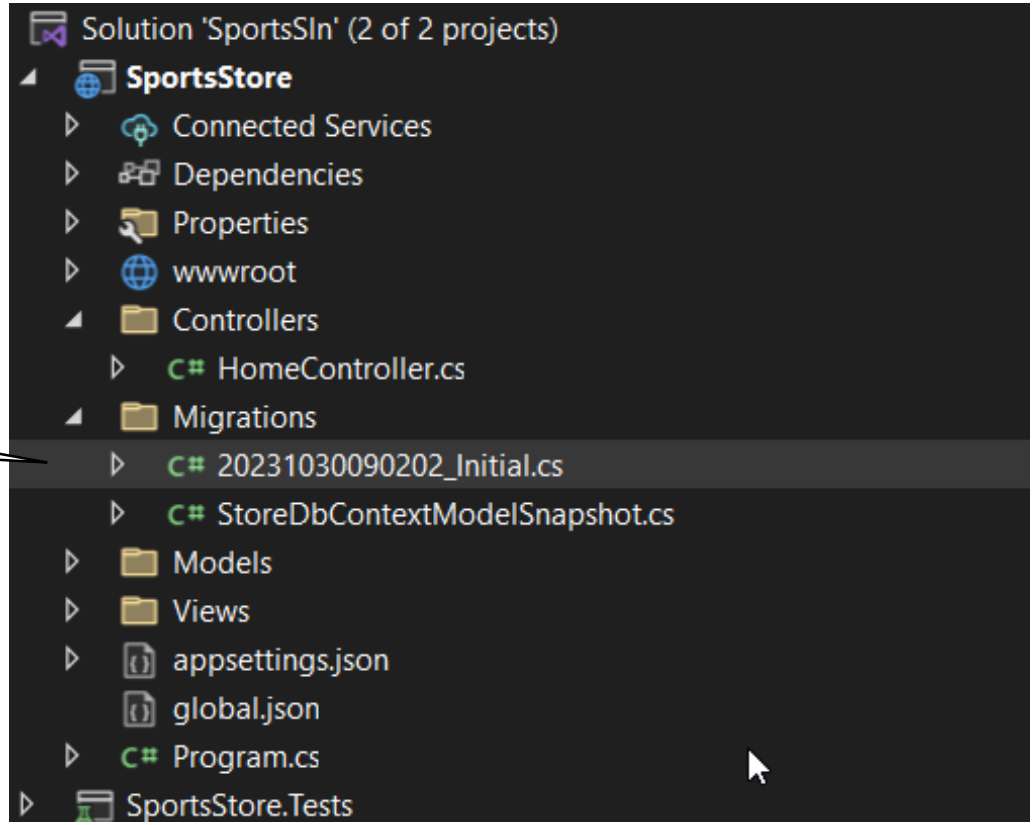
```
using Microsoft.EntityFrameworkCore;
using SportsStore.Models;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<StoreDbContext>(opts => {
    opts.UseSqlServer(
        builder.Configuration["ConnectionStrings:SportsStoreConnection"]);
});
builder.Services.AddScoped<IStoreRepository, EFStoreRepository>();
var app = builder.Build();
//app.MapGet('/', () => 'Hello World!');
app.UseStaticFiles();
app.MapDefaultControllerRoute();
app.Run();
```

2.6 Creating the database migration

- Power shell:

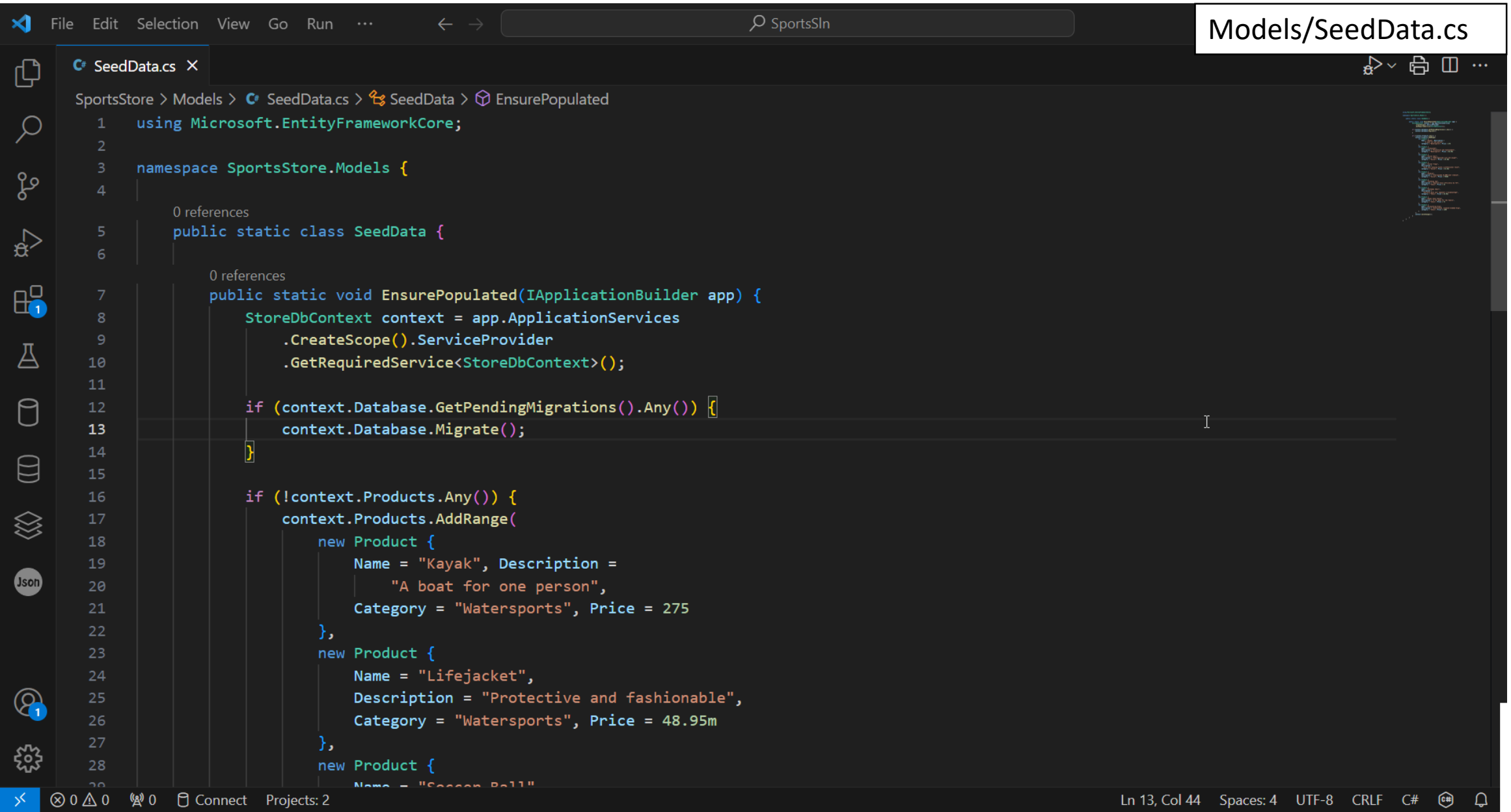
```
dotnet ef migrations add Initial
```

Thư mục Migrations: chứa các class migration



2.7 Creating seed data

Models/SeedData.cs



Seeding the database in Program.cs

Program.cs

```
using Microsoft.EntityFrameworkCore;
using SportsStore.Models;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<StoreDbContext>(opts => {
    opts.UseSqlServer(
        builder.Configuration["ConnectionStrings:SportsStoreConnection"]);
});
builder.Services.AddScoped<IStoreRepository, EFStoreRepository>();
var app = builder.Build();
//app.MapGet('/', () => 'Hello World!');
app.UseStaticFiles();
app.MapDefaultControllerRoute();
SeedData.EnsurePopulated(app);
app.Run();
```

Resetting the database

- Power shell :

```
dotnet ef database drop --force --context StoreDbContext
```

CSDL SportsStore sẽ tạo lại với dữ liệu được khai báo trong file SeedData.cs

3 Displaying a list of products

- Preparing the controller: HomeController.cs
- Writing Unit test: Can_User_Repository
- Updating the view : Index.cshtml
- Run application : http://localhost:5000

3.1 Preparing the controller

Controllers/HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using SportsStore.Models;

namespace SportsStore.Controllers {
    public class HomeController : Controller {
        private IStoreRepository repository;

        public HomeController(IStoreRepository repo) {
            repository = repo;
        }

        public IActionResult Index() => View(repository.Products);
    }
}
```

Unit test: repository access

SportsStore.Test/HomeControllerTests.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.AspNetCore.Mvc;
using Moq;
using SportsStore.Controllers;
using SportsStore.Models;
using Xunit;

namespace SportsStore.Tests {
    public class HomeControllerTests {
        [Fact]
        public void Can_Use_Repository() {
            // Arrange
            Mock<IStoreRepository> mock = new Mock<IStoreRepository>();
            mock.Setup(m => m.Products).Returns((new Product[] {
                new Product {ProductID = 1, Name = "P1"},
                new Product {ProductID = 2, Name = "P2"}
            }).AsQueryable<Product>());
            HomeController controller = new HomeController(mock.Object);
            // Act
            IEnumerable<Product>? result =
                (controller.Index() as ViewResult)?.ViewData.Model as IEnumerable<Product>;
            // Assert
            Product[] prodArray = result?.ToArray() ?? Array.Empty<Product>();
            Assert.True(prodArray.Length == 2);
            Assert.Equal("P1", prodArray[0].Name);
            Assert.Equal("P2", prodArray[1].Name);
        }
    }
}
```

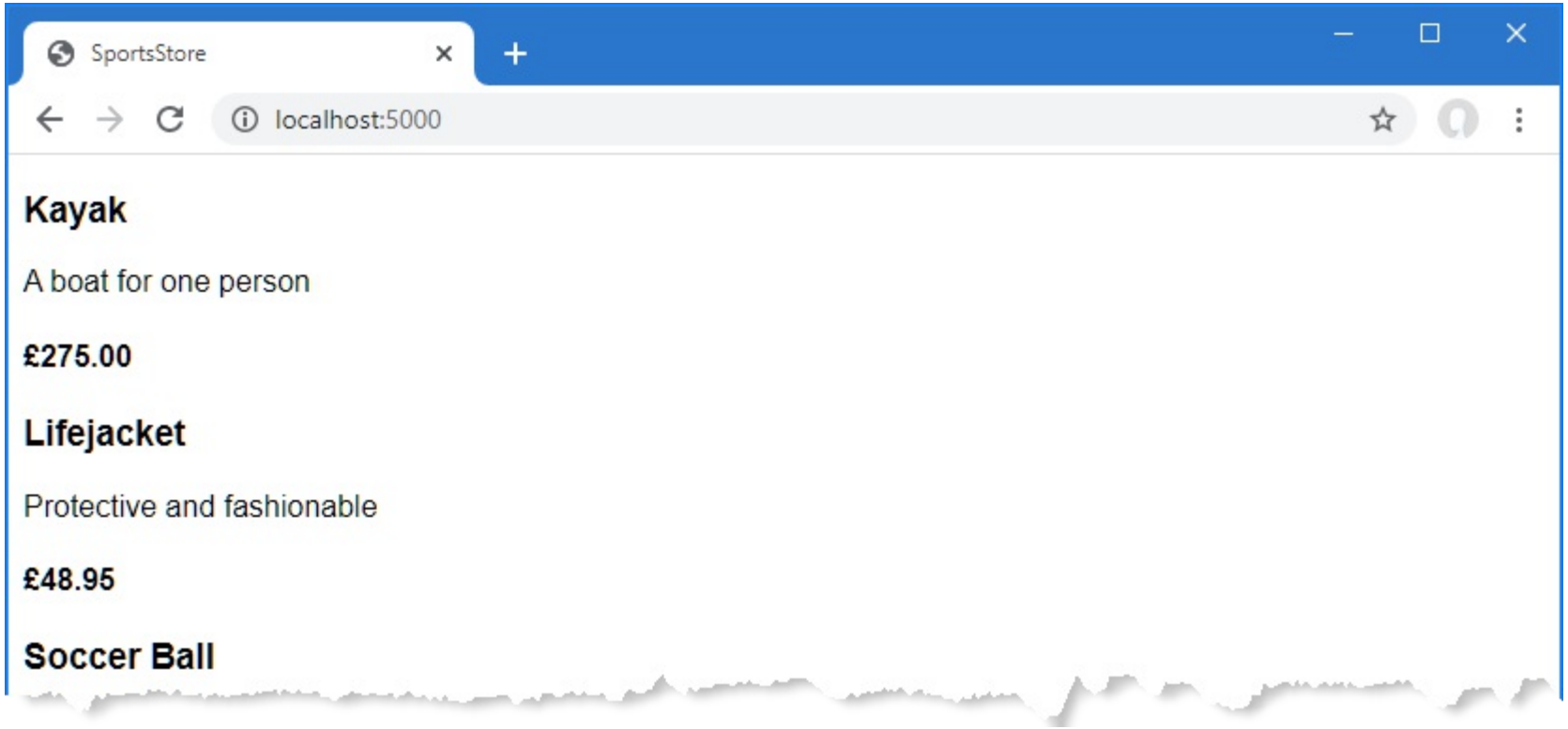
3.2 Updating the view

Views/Home/Index.cshtml

```
@model IQueryable<Product>

@foreach (var p in Model ?? Enumerable.Empty<Product>()) {
    <div>
        <h3>@p.Name</h3>
        @p.Description
        <h4>@p.Price.ToString("c")</h4>
    </div>
}
```

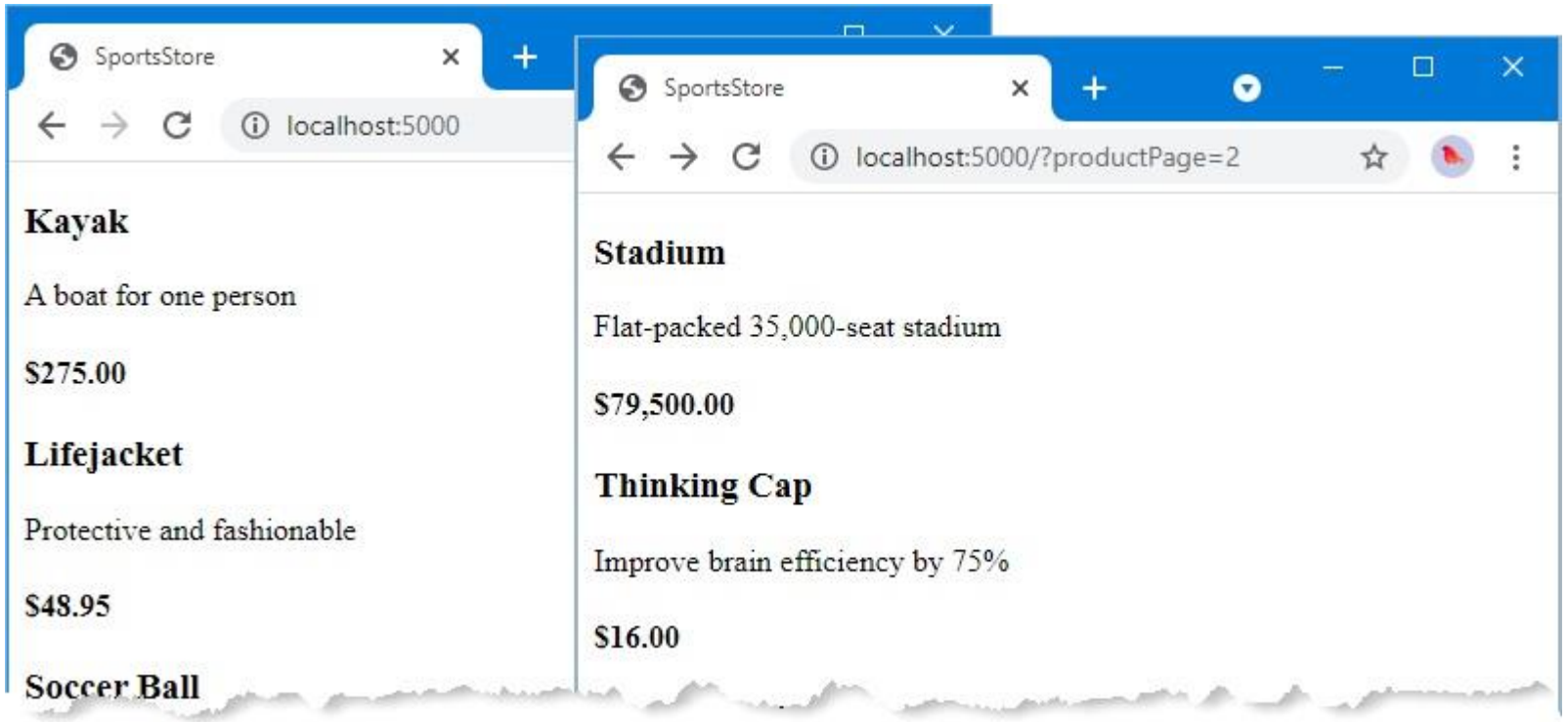
3.3 Running the application

- Request: http://localhost:5000



4 Adding pagination

- Http request: http://localhost:5000
- Http request: http://localhost:5000/?productPage=2



Adding pagination in the HomeController.cs

Controllers/HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using SportsStore.Models;
namespace SportsStore.Controllers {
    public class HomeController : Controller {
        private IStoreRepository repository;
        public int PageSize = 4;
        public HomeController(IStoreRepository repo) {
            repository = repo;
        }
        public ViewResult Index(int productPage = 1)
            => View(repository.Products
                .OrderBy(p => p.ProductID)
                .Skip((productPage - 1) * PageSize)
                .Take(PageSize));
    }
}
```

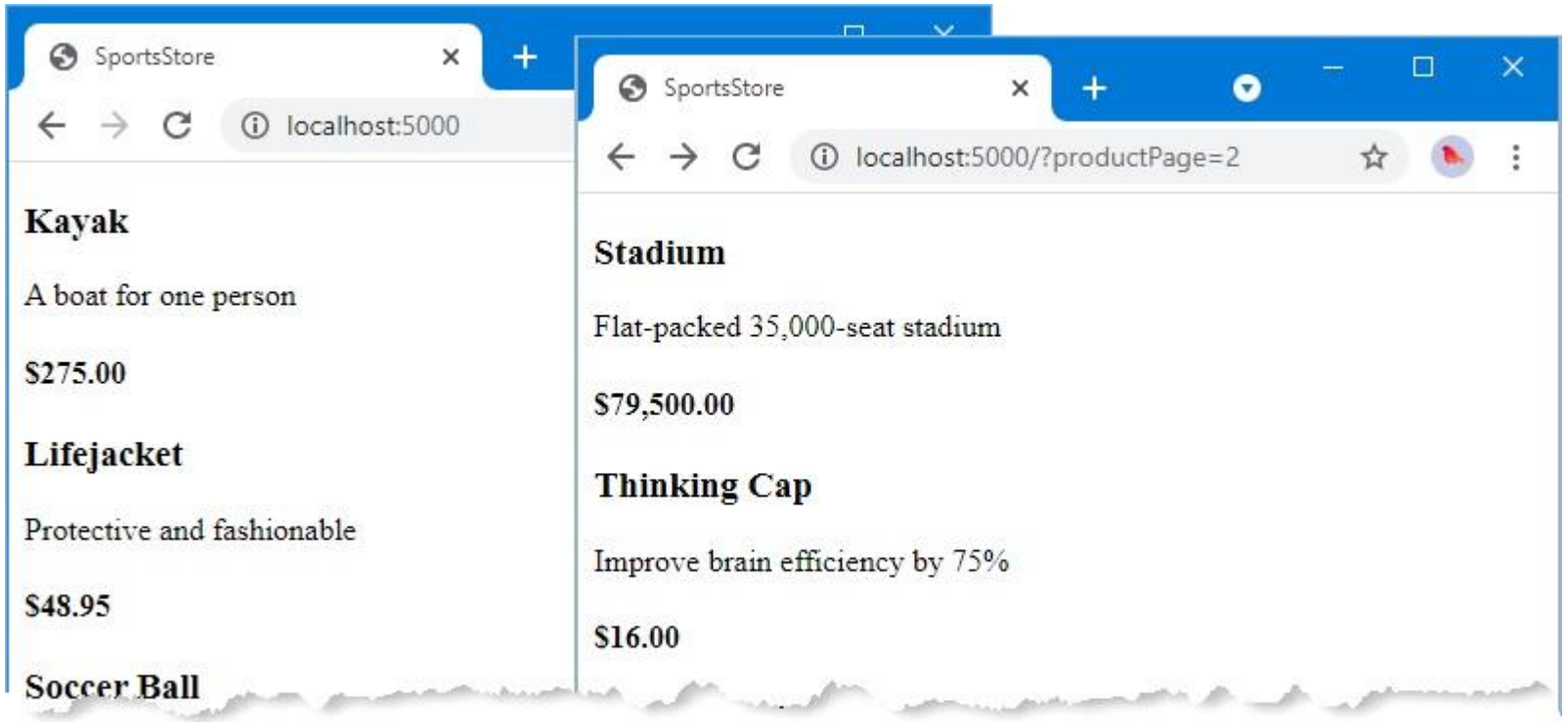
Unit Test: Pagination

SportsStore.Tests/HomcontrolerTests.cs

```
[Fact]
public void Can_Paginate() {
    // Arrange
    Mock<IStoreRepository> mock = new Mock<IStoreRepository>();
    mock.Setup(m => m.Products).Returns((new Product[] {
        new Product {ProductID = 1, Name = 'P1'},
        new Product {ProductID = 2, Name = 'P2'},
        new Product {ProductID = 3, Name = 'P3'},
        new Product {ProductID = 4, Name = 'P4'},
        new Product {ProductID = 5, Name = 'P5'}
    }).AsQueryable<Product>());
    HomeController controller = new HomeController(mock.Object);
    controller.PageSize = 3;
    // Act
    IEnumerable<Product> result =
        (controller.Index(2) as ViewResult)?.ViewData.Model
            as IEnumerable<Product>
                ?? Enumerable.Empty<Product>();
    // Assert
    Product[] prodArray = result.ToArray();
    Assert.True(prodArray.Length == 2);
    Assert.Equal('P4', prodArray[0].Name);
    Assert.Equal('P5', prodArray[1].Name);
}
```

4.1 Displaying Page Links

- Http request: http://localhost:5000
- Http request: http://localhost:5000/?productPage=2



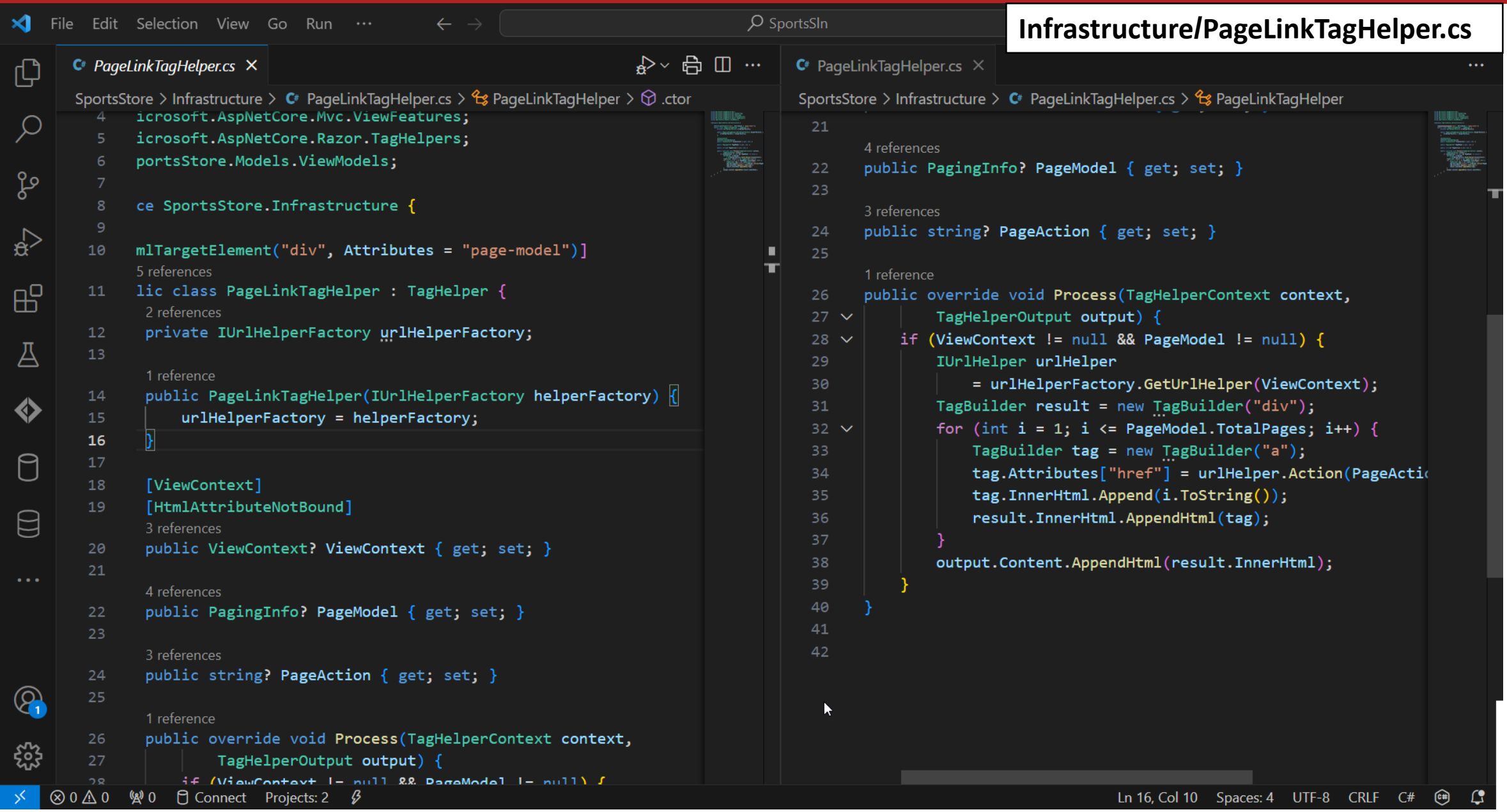
Adding Pagination

Models/ViewModels/PagingInfo.cs

```
namespace SportsStore.Models.ViewModels {

    public class PagingInfo {
        public int TotalItems { get; set; }
        public int ItemsPerPage { get; set; }
        public int CurrentPage { get; set; }

        public int TotalPages =>
            (int)Math.Ceiling((decimal)TotalItems / ItemsPerPage);
    }
}
```

Adding the tag helper class

Infrastructure/PageLinkTagHelper.cs



Registering a tag helper in the _ViewImports.cshtml file

Views/ _ViewImports.cshtml

```
@using SportsStore.Models
@using SportsStore.Models.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, SportsStore
```

UNIT TEST: creating page links

SportsStore.Tests/PageLinkTagHelperTests.cs

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Razor.TagHelpers;
using Moq;
using SportsStore.Infrastructure;
using SportsStore.Models.ViewModels;
using Xunit;

namespace SportsStore.Tests {

    0 references
    public class PageLinkTagHelperTests {

        [Fact]
        0 references
        public void Can_Generate_Page_Links() {
            // Arrange
            var urlHelper = new Mock<IUrlHelper>();
            urlHelper.SetupSequence(x =>
                    x.Action(It.IsAny<UrlActionContext>()))
                .Returns("Test/Page1")
                .Returns("Test/Page2")
                .Returns("Test/Page3");

            var urlHelperFactory = new Mock<IUrlHelperFactory>(
            urlHelperFactory.Setup(f =>
                    f.GetUrlHelper(It.IsAny<ActionContext>()))
```

```
                        CurrentPage = 2,
                        TotalItems = 28,
                        ItemsPerPage = 10
                    },
                    ViewContext = viewContext.Object,
                    PageAction = "Test"
                };

            TagHelperContext ctx = new TagHelperContext(
                new TagHelperAttributeList(),
                new Dictionary<object, object>(), "");

            var content = new Mock<TagHelperContent>();
            TagHelperOutput output = new TagHelperOutput("div",
                new TagHelperAttributeList(),
                (cache, encoder) => Task.FromResult(content.Obj

            // Act
            helper.Process(ctx, output);

            // Assert
            Assert.Equal(@"<a href=""Test/Page1"">1</a>"
                + @"<a href=""Test/Page2"">2</a>"
                + @"<a href=""Test/Page3"">3</a>",
                output.Content.GetContent());
        }
    }
}
```

Adding the View Model data

Models/ViewModels/ProductListViewModel.cs

```
namespace SportsStore.Models.ViewModels {

    public class ProductsListViewModel {
        public IEnumerable<Product> Products { get; set; } = Enumerable.Empty<Product>();
        public PagingInfo PagingInfo { get; set; } = new();
    }
}
```

Updating action method in the HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
using SportsStore.Models;
using SportsStore.Models.ViewModels;

namespace SportsStore.Controllers {
    public class HomeController : Controller {
        private IStoreRepository repository;
        public int PageSize = 4;

        public HomeController(IStoreRepository repo) {
            repository = repo;
        }

        public ViewResult Index(int productPage = 1)
           => View(new ProductsListViewModel {
               Products = repository.Products
                   .OrderBy(p => p.ProductID)
                   .Skip((productPage - 1) * PageSize)
                   .Take(PageSize),
               PagingInfo = new PagingInfo {
                   CurrentPage = productPage,
                   ItemsPerPage = PageSize,
                   TotalItems = repository.Products.Count()
               }
           });
    }
}
```

UNIT TEST: page model view data

SportsStore.Tests/HomeControllerTests.cs

```
[Fact]
public void Can_Send_Pagination_View_Model() {
    // Arrange
    Mock<IStoreRepository> mock = new Mock<IStoreRepository>();
    mock.Setup(m => m.Products).Returns((new Product[] {
        new Product {ProductID = 1, Name = "P1"},
        new Product {ProductID = 2, Name = "P2"},
        new Product {ProductID = 3, Name = "P3"},
        new Product {ProductID = 4, Name = "P4"},
        new Product {ProductID = 5, Name = "P5"}
    }).AsQueryable<Product>());
    // Arrange
    HomeController controller =
        new HomeController(mock.Object) { PageSize = 3 };
    // Act
    ProductsListViewModel result =
        controller.Index(2)?.ViewData.Model as ProductsListViewModel ?? new();
    // Assert
    PagingInfo pageInfo = result.PagingInfo;
    Assert.Equal(2, pageInfo.CurrentPage);
    Assert.Equal(3, pageInfo.ItemsPerPage);
```

UNIT TEST: page model view data

SportsStore.Tests/HomeControllerTests.cs

```
public void Can_Use_Repository() {
    // Arrange
    Mock<IStoreRepository> mock = new Mock<IStoreRepository>();
    mock.Setup(m => m.Products).Returns((new Product[] {
        new Product {ProductID = 1, Name = "P1"},
        new Product {ProductID = 2, Name = "P2"}
    }).AsQueryable<Product>());
    HomeController controller = new HomeController(mock.Object);
    // Act
    // IEnumerable<Product>? result =
    //     (controller.Index() as ViewResult)?.ViewData.Model as IEnumerable<Product>;
    ProductsListViewModel result =
    controller.Index()?.ViewData.Model as ProductsListViewModel ?? new();
    // Assert
    // Product[] prodArray = result?.ToArray() ?? Array.Empty<Product>();
    Product[] prodArray = result.Products.ToArray();
    Assert.True(prodArray.Length == 2);
    Assert.Equal("P1", prodArray[0].Name);
    Assert.Equal("P2", prodArray[1].Name);
}
```

UNIT TEST: page model view data

SportsStore.Tests/HomeControllerTests.cs

```
[Fact]
 public void Can_Paginate() {
     // Arrange
     Mock<IStoreRepository> mock = new Mock<IStoreRepository>();
     mock.Setup(m => m.Products).Returns((new Product[] {
         new Product {ProductID = 1, Name = "P1"},
         new Product {ProductID = 2, Name = "P2"},
         new Product {ProductID = 3, Name = "P3"},
         new Product {ProductID = 4, Name = "P4"},
         new Product {ProductID = 5, Name = "P5"}
     }).AsQueryable<Product>());
     HomeController controller = new HomeController(mock.Object);
     controller.PageSize = 3;
     // Act
     // IEnumerable<Product> result =
     //     (controller.Index(2) as ViewResult)?.ViewData.Model
     //         as IEnumerable<Product>
     //                 ?? Enumerable.Empty<Product>();
     ProductsListViewModel result =
     controller.Index(2)?.ViewData.Model as ProductsListViewModel ?? new();
     // Assert
     // Product[] prodArray = result.ToArray();
     Product[] prodArray = result.Products.ToArray();
     Assert.True(prodArray.Length == 2);
     Assert.Equal("P4", prodArray[0].Name);
     Assert.Equal("P5", prodArray[1].Name);
 }
```

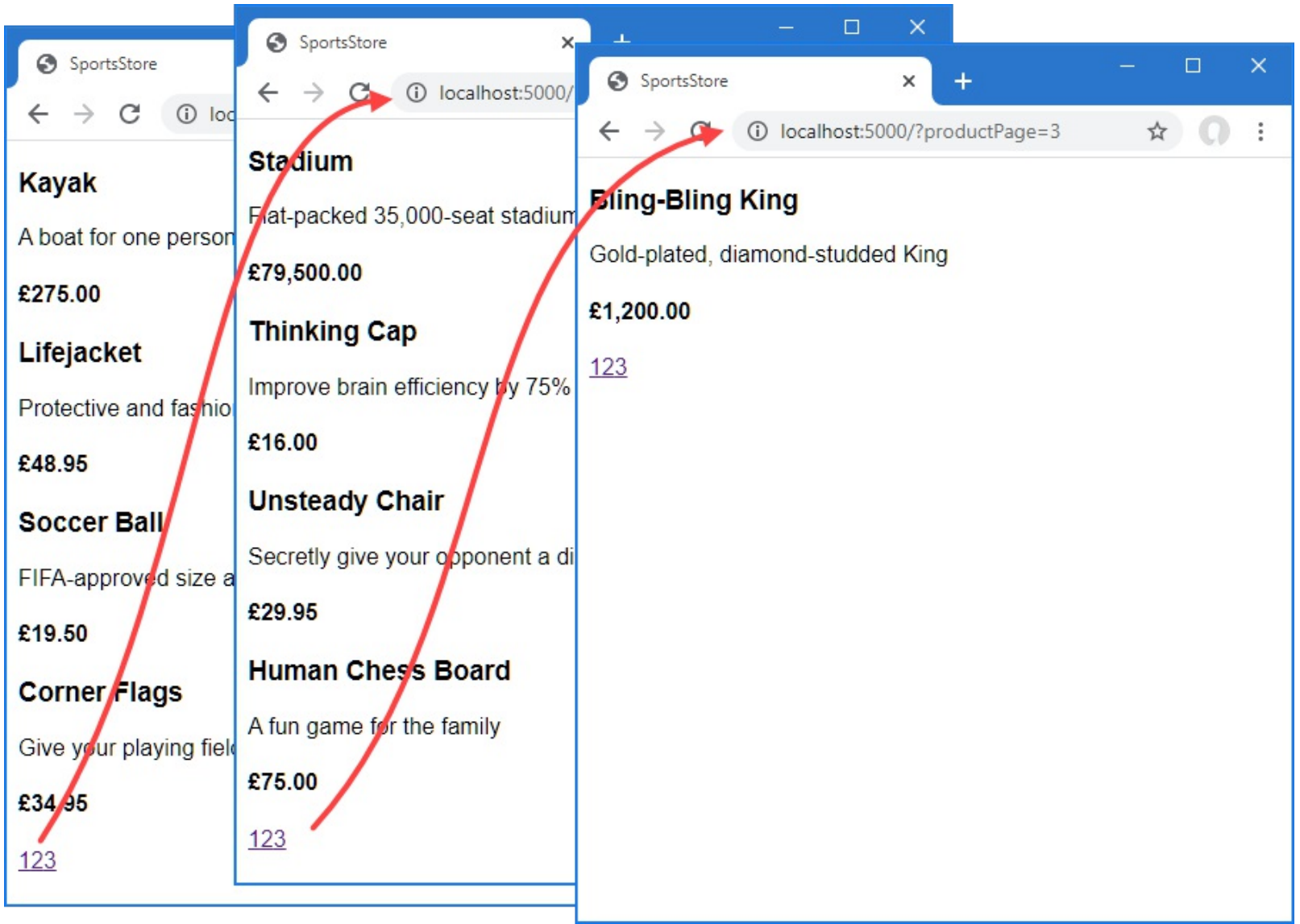
Updating the Index.cshtml

Views/Home/Index.cshtml

```
@model ProductsListViewModel

@foreach (var p in Model?.Products ?? Enumerable.Empty<Product>()) {
    <div>
        <h3>@p.Name</h3>
        @p.Description
        <h4>@p.Price.ToString("c")</h4>
    </div>
}
<div page-model="@Model?.PagingInfo" page-action="Index"></div>
```

Run application

- Request: http://localhost:5000



4.2 Improving the URLs

- Từ http request http://localhost/?productPage=2 chuyển thành http request http://localhost/Page2
- Tạo định tuyến (route) mới trong file `Program.cs`

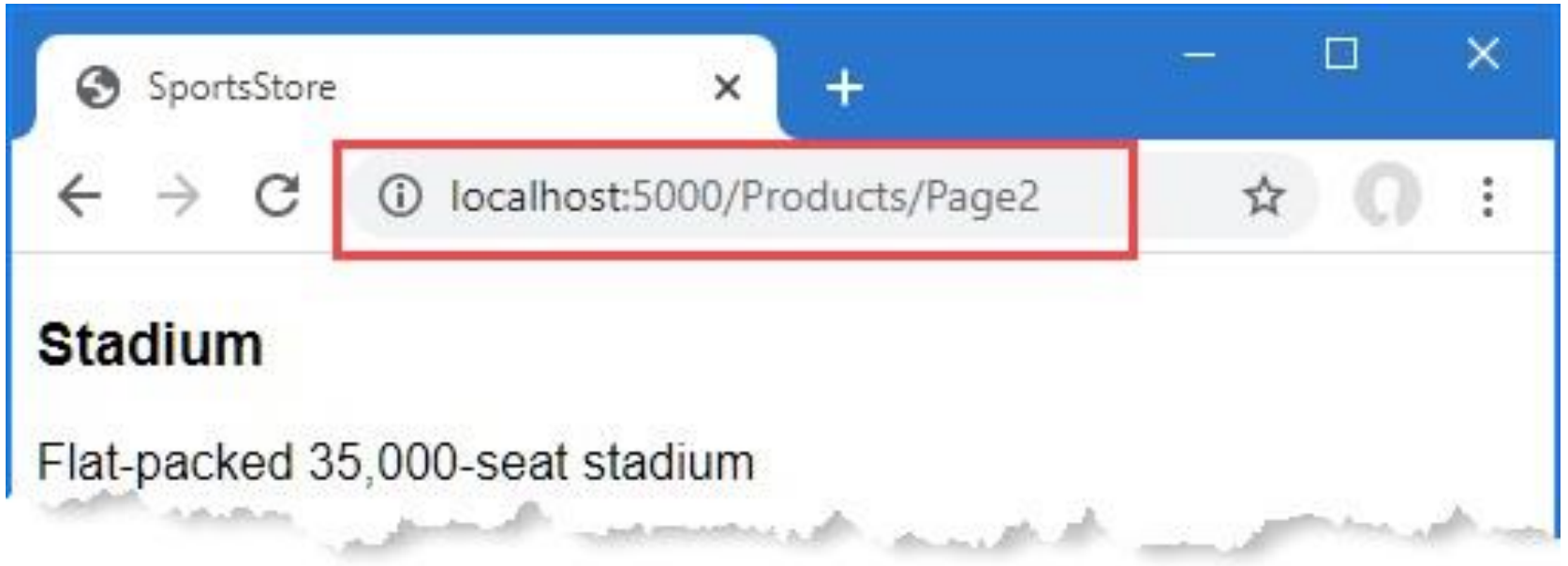
Adding a New Route in the Program.cs file

Program.cs

```
using Microsoft.EntityFrameworkCore;
using SportsStore.Models;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<StoreDbContext>(opts => {
    opts.UseSqlServer(
        builder.Configuration["ConnectionStrings:SportsStoreConnection"]);
});
builder.Services.AddScoped<IStoreRepository, EFStoreRepository>();
var app = builder.Build();
//app.MapGet('/', () => 'Hello World!');
app.UseStaticFiles();
app.MapControllerRoute("pagination",
    "Products/Page{productPage}",
    new { Controller = "Home", action = "Index" });
app.MapDefaultControllerRoute();

SeedData.EnsurePopulated(app);
app.Run();
```

Run Application

- Http request: http://localhost:5000 → click pagination link → http request http://localhost/Page2



5 Styling the content

- Thiết kế layout cho ứng dụng SportsStore

5.1 Installing the Bootstrap package

- Installing the LibMan tool package

```
dotnet tool uninstall --global Microsoft.Web.LibraryManager.Cli
dotnet tool install --global Microsoft.Web.LibraryManager.Cli --version 2.1.175
```

- Initializing the example project

```
libman init -p cdnjs
libman install bootstrap@5.2.3 -d wwwroot/lib/bootstrap
```

5.2 Aplying Bootstrap styles

Views/Shared/_Layout.cshtml

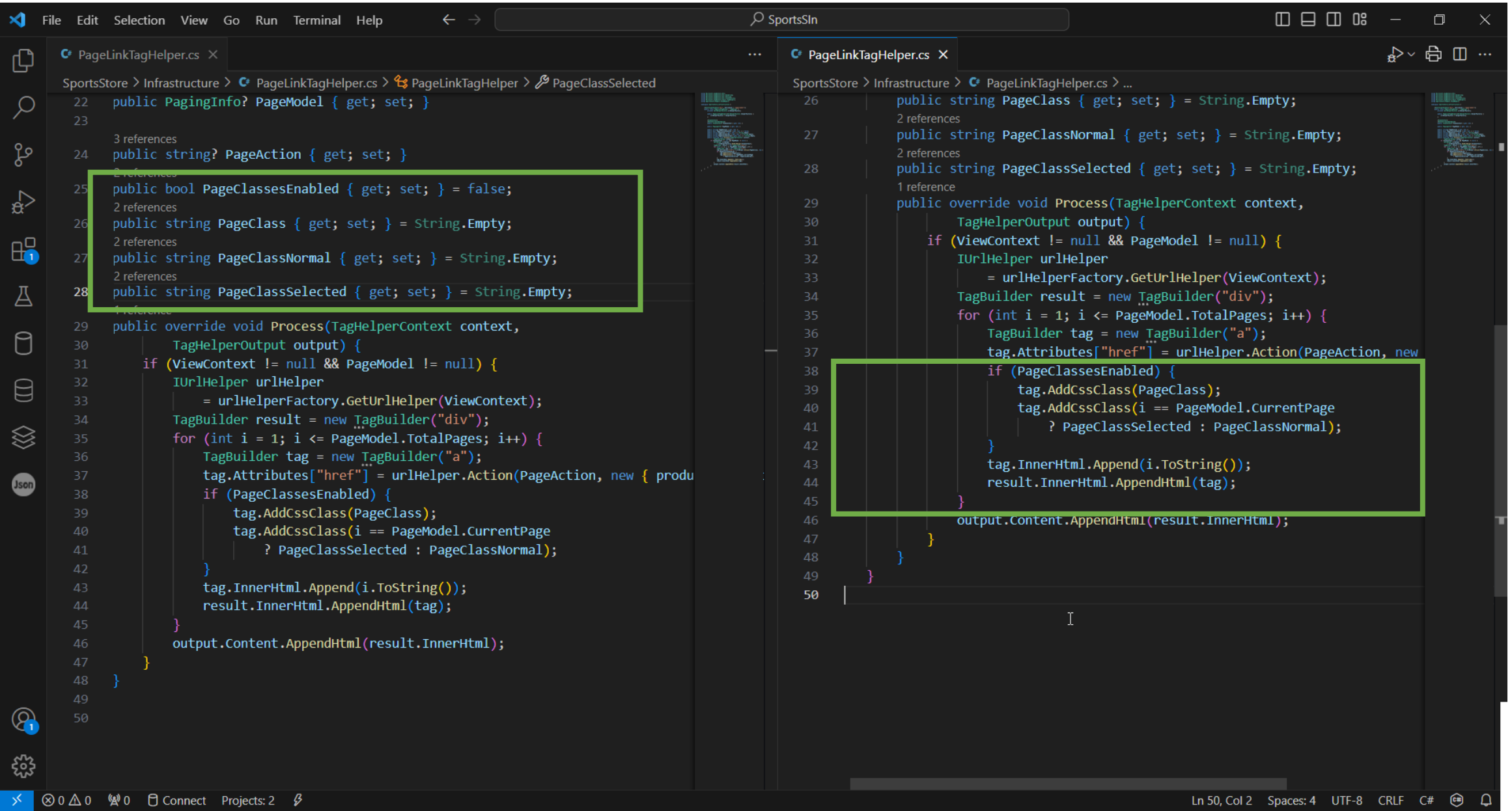
```
<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>SportsStore</title>
    <link href="/lib/bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
    <div class="bg-dark text-white p-2">
        <span class="navbar-brand ml-2">SPORTS STORE</span>
    </div>
    <div class="row m-1 p-1">
        <div id="categories" class="col-3">
            Put something useful here later
        </div>
        <div class="col-9">
            @RenderBody()
        </div>
    </div>
</body>
</html>
```

Styling Content in the Index.cshtml

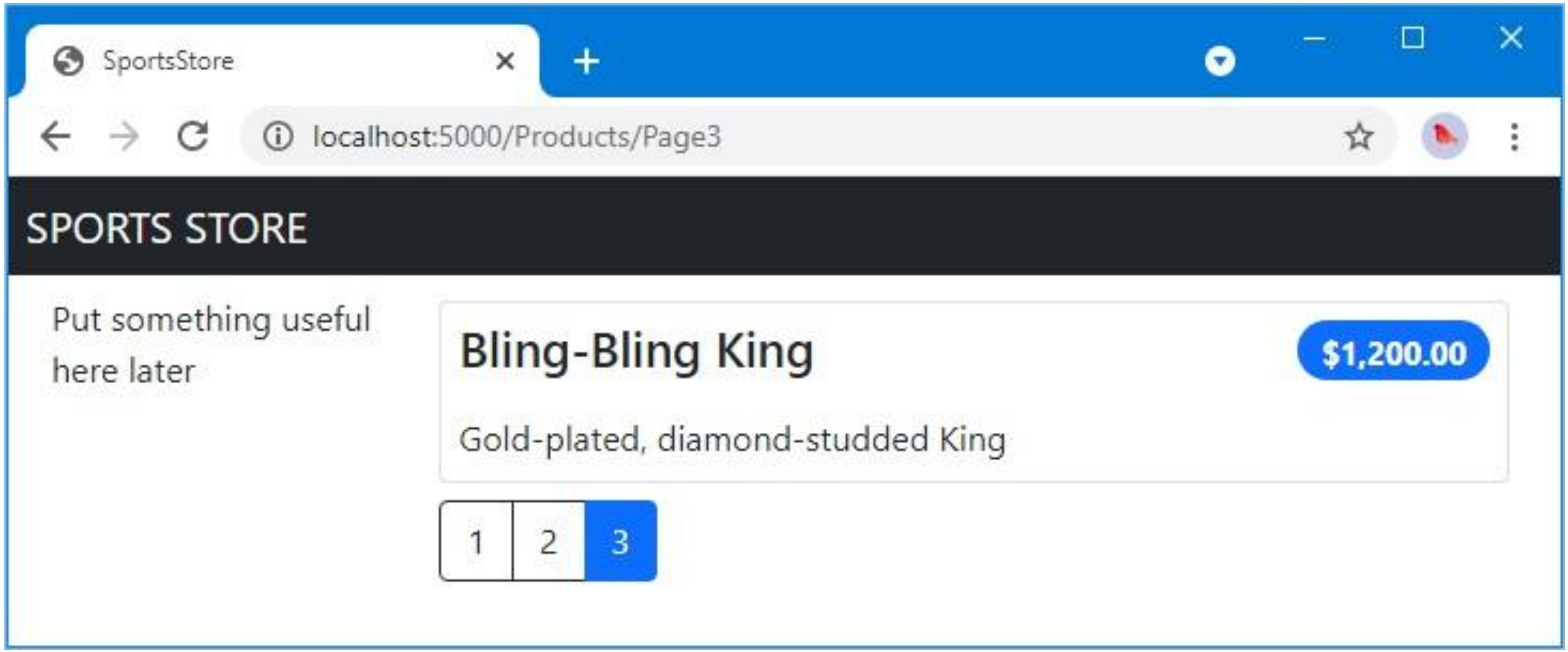
Views/Home/Index.cshtml

```
@model ProductsListViewModel
@foreach (var p in Model?.Products ?? Enumerable.Empty<Product>()) {
    <div class="card card-outline-primary m-1 p-1">
        <div class="bg-faded p-1">
            <h4>
                @p.Name
                <span class="badge rounded-pill bg-primary text-white"
                        style="float:right">
                    <small>@p.Price.ToString("c")</small>
                </span>
            </h4>
        </div>
        <div class="card-text p-1">@p.Description</div>
    </div>
}
<div page-model="@Model?.PagingInfo" page-action="Index" page-classes-enabled="true"
     page-class="btn" page-class-normal="btn-outline-dark"
     page-class-selected="btn-primary" class="btn-group pull-right m-1">
</div>
```

Adding Classes to Elements in the PageLinkTagHelper.cs



Run SportsStore App



5.3 Creating a partial view

- Tạo parital view để tích hợp nội dung vào view khác
- Giảm sự trùng lặp code khi cần hiển thị nội dung ở nhiều view khác nhau
- Thay vì copy và paste các trang Razor vào các view khác → chỉ cần định nghĩa một lần trong partial view.

Creating partial view ProductsSummary.cshtml

Views/Shared/ProductSummary.cshtml

```
@model Product

<div class="card card-outline-primary m-1 p-1">
    <div class="bg-faded p-1">
        <h4>
            @Model.Name
            <span class="badge rounded-pill bg-primary text-white"
                  style="float:right">
                <small>@Model.Price.ToString("c")</small>
            </span>
        </h4>
    </div>
    <div class="card-text p-1">@Model.Description</div>
</div>
```

Using Partial View in the Index.cshtml file

Views/Index.cshtm

```
@model ProductsListViewModel
@foreach (var p in Model?.Products ?? Enumerable.Empty<Product>()) {
    <partial name="ProductSummary" model="p" />
}
<div page-model="@Model?.PagingInfo" page-action="Index" page-classes-enabled="true"
     page-class="btn" page-class-normal="btn-outline-dark"
     page-class-selected="btn-primary" class="btn-group pull-right m-1">
</div>
```

- Run SportsStore App
- Request : http://localhost:5000