

SportsStore – A real application

Bài 03: Part 1

²Nội 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

³1 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
```

⁴1.1 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
```

⁵1.2 Opening the project

- Visual Studio 2022
- Visual Studio Code

⁶1.3 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"  
      }  
    }  
  }  
}
```

⁷1.4 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.

⁸1.5 Preparing the service and the request pipeline

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

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

builder.Services.AddControllersWithViews();

var app = builder.Build();

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

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

app.Run();
```

Program.cs

⁹ 1.6 Configuring the Razor view engine

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

Views/_ViewImports.cshtml

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

Views/_ViewStart.cshtml

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

Views/Shared/_Layout.cshtml

1.7 Creating the controller and view

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

namespace SportsStore.Controllers

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

Controllers/HomeController.cs

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

Views/Index.cshtml

¹¹1.8 Starting the data model

```
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;

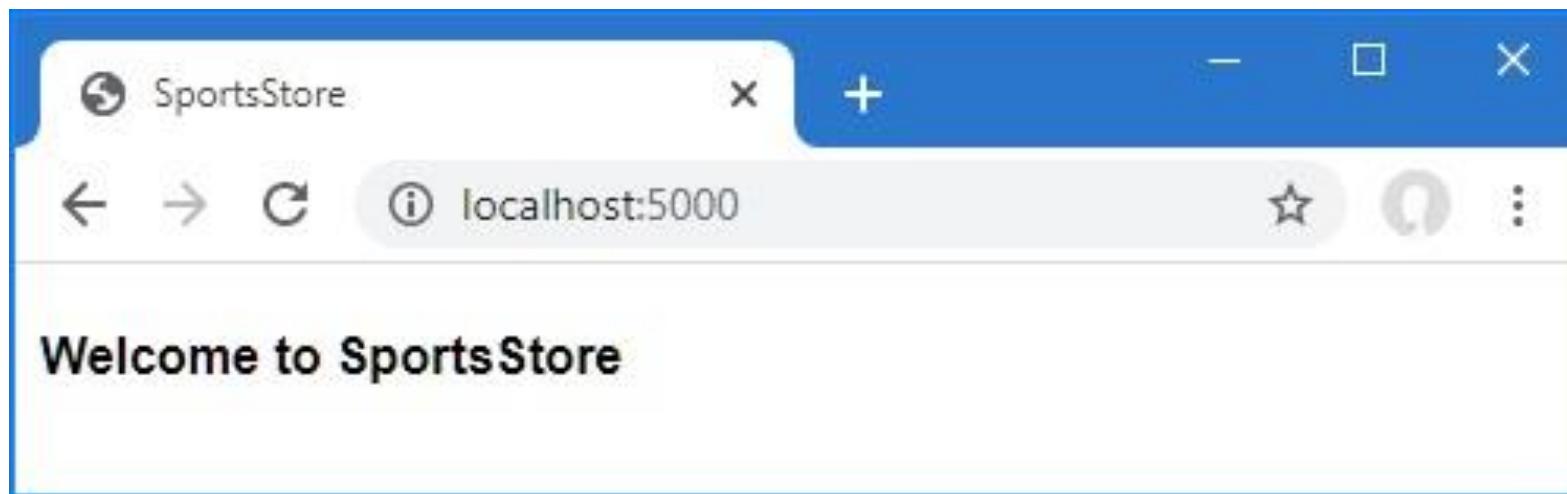
        [Column(TypeName = "decimal(8, 2)")]
        public decimal Price { get; set; }

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

Model/Product.cs

¹²1.9 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

¹⁴2.1 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

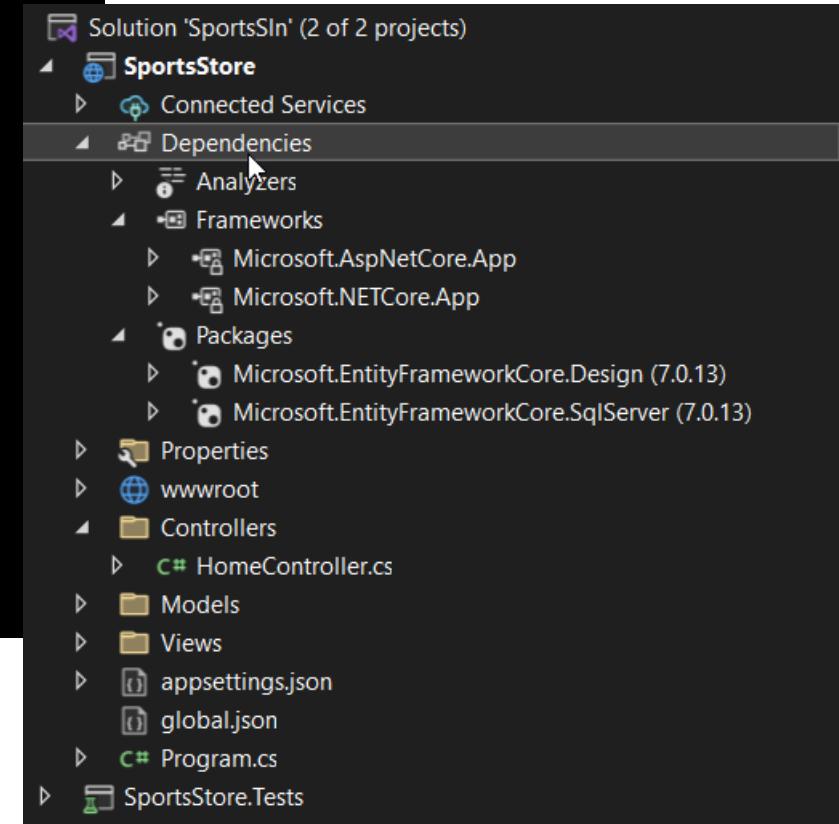
```
dotnet tool uninstall --global dotnet-ef  
dotnet tool install --global dotnet-ef --version 7.0.13
```

2.1 Installing the Entity Framework Core package

- Thông tin Entity Framework và package trong SportsStore project

```
<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>
```

SportsStore.csproj



2.2 Defining the connection string

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

appsettings.json

¹⁷2.3 Creating the database context class

```
using Microsoft.EntityFrameworkCore;
```

Models/StoreDbContext.cs

```
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

```
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();
```

Program.cs

2.5 Creating a repository

```
namespace SportsStore.Models {  
    public interface IStoreRepository {  
  
        IQueryable<Product> Products { get; }  
    }  
}
```

Models/IStoreRepository.cs

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

Models/EFStoreRepository.cs

2.5 Creating a repository

```
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();
```

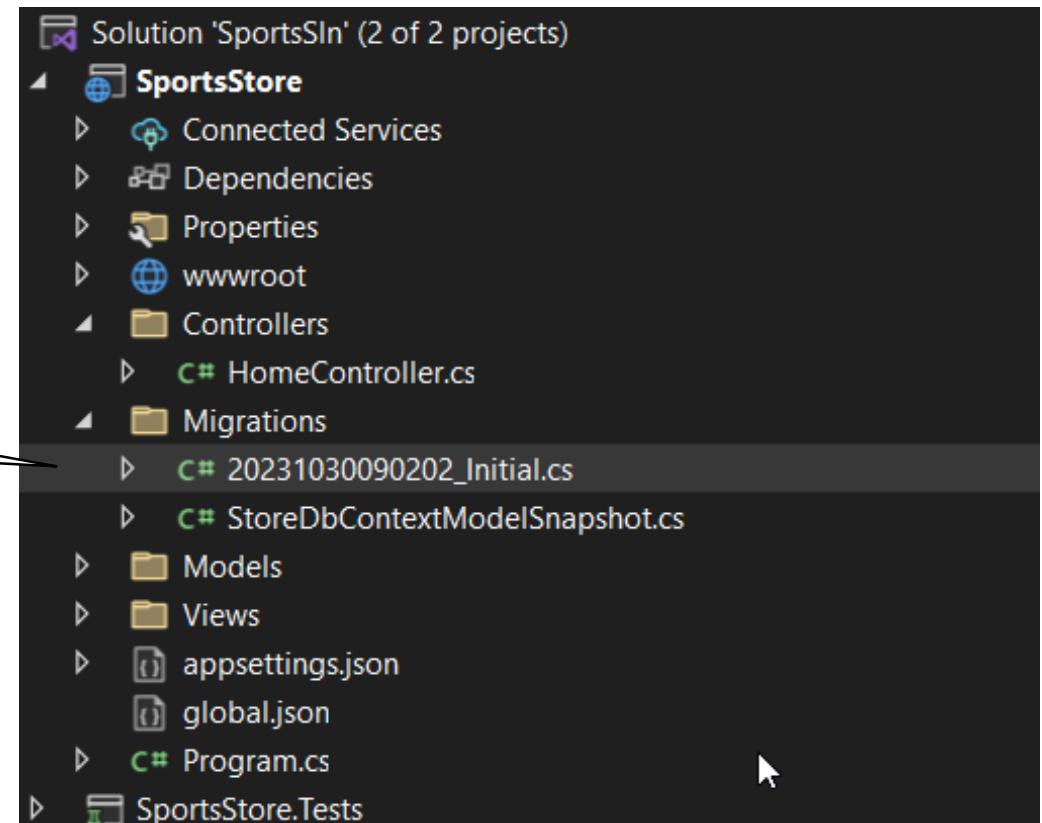
Program.cs

2.6 Creating the database migration

- Power shell:

```
dotnet ef migrations add Initial
```

Thư mục Migrations: chưa có các class migration



2.7 Creating seed data

The screenshot shows a Visual Studio interface with the following details:

- Title Bar:** SportsSln
- File Explorer:** Shows the project structure: SportsStore > Models > SeedData.cs.
- Solution Explorer:** Shows 1 item in the Models folder.
- Toolbox:** Standard Visual Studio icons for File, Edit, Selection, View, Go, Run, etc.
- Code Editor:** The file `SeedData.cs` is open, showing C# code for seeding a database with products. The code includes migrations and adds ranges of products like Kayak, Lifejacket, and Soccer Ball.
- Status Bar:** Ln 13, Col 44, Spaces: 4, UTF-8, C#, etc.

```
1  using Microsoft.EntityFrameworkCore;
2
3  namespace SportsStore.Models {
4
5      public static class SeedData {
6
7          public static void EnsurePopulated(IApplicationBuilder app) {
8              StoreDbContext context = app.ApplicationServices
9                  .CreateScope().ServiceProvider
10                 .GetRequiredService<StoreDbContext>();
11
12             if (context.Database.GetPendingMigrations().Any()) {
13                 context.Database.Migrate();
14             }
15
16             if (!context.Products.Any()) {
17                 context.Products.AddRange(
18                     new Product {
19                         Name = "Kayak", Description =
20                             "A boat for one person",
21                         Category = "Watersports", Price = 275
22                     },
23                     new Product {
24                         Name = "Lifejacket",
25                         Description = "Protective and fashionable",
26                         Category = "Watersports", Price = 48.95m
27                     },
28                     new Product {
29                         Name = "Soccer Ball"
30                     }
31                 );
32             }
33         }
34     }
35 }
```

Seeding the database in 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();
```

Program.cs

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

```
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);
    }
}
```

Controllers/HomeController.cs

Unit test: repository access

```
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);
        }
    }
}
```

SportsStore.Test/HomeControllerTests.cs

3.2 Updating the view

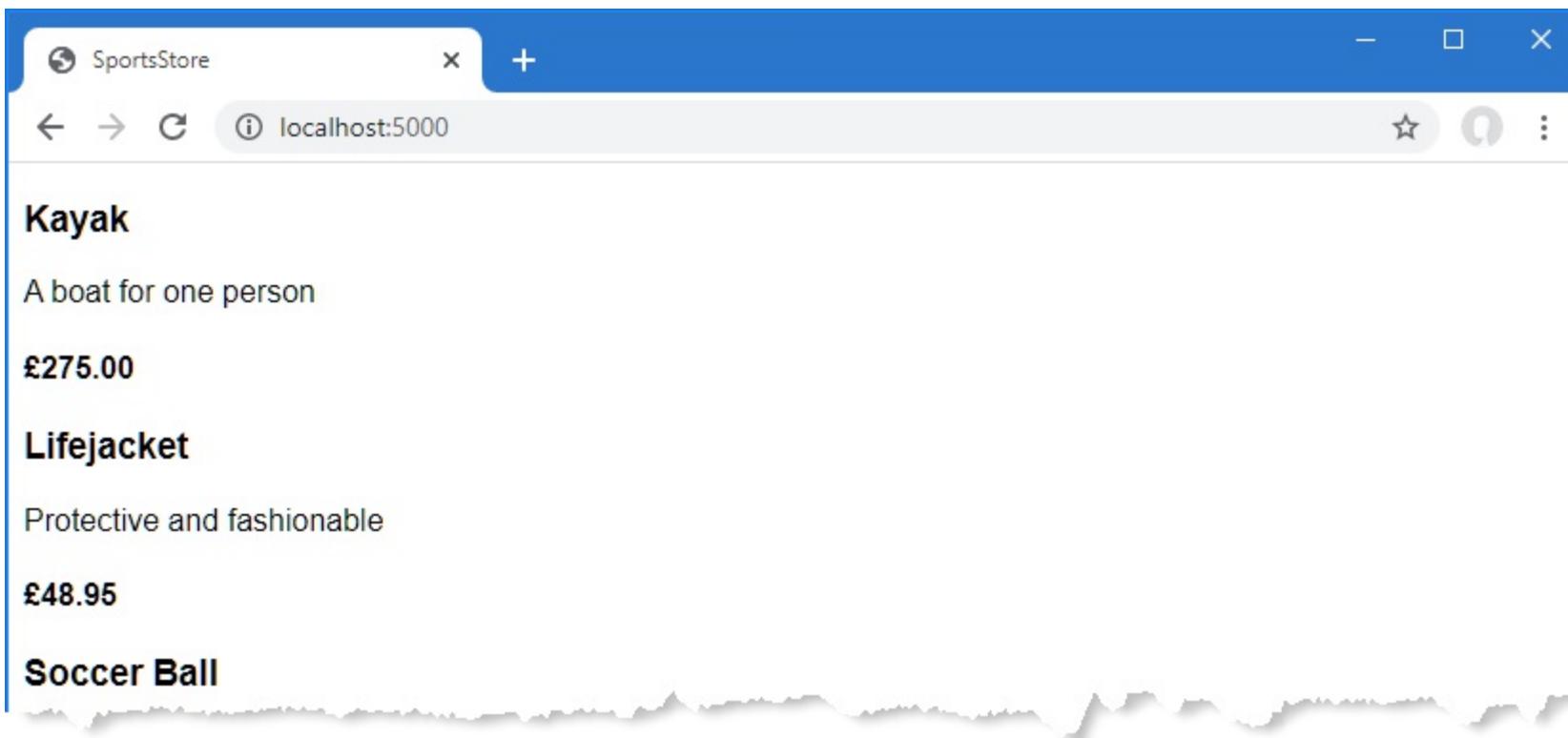
```
@model IQueryable<Product>
```

Views/Home/Index.cshtml

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

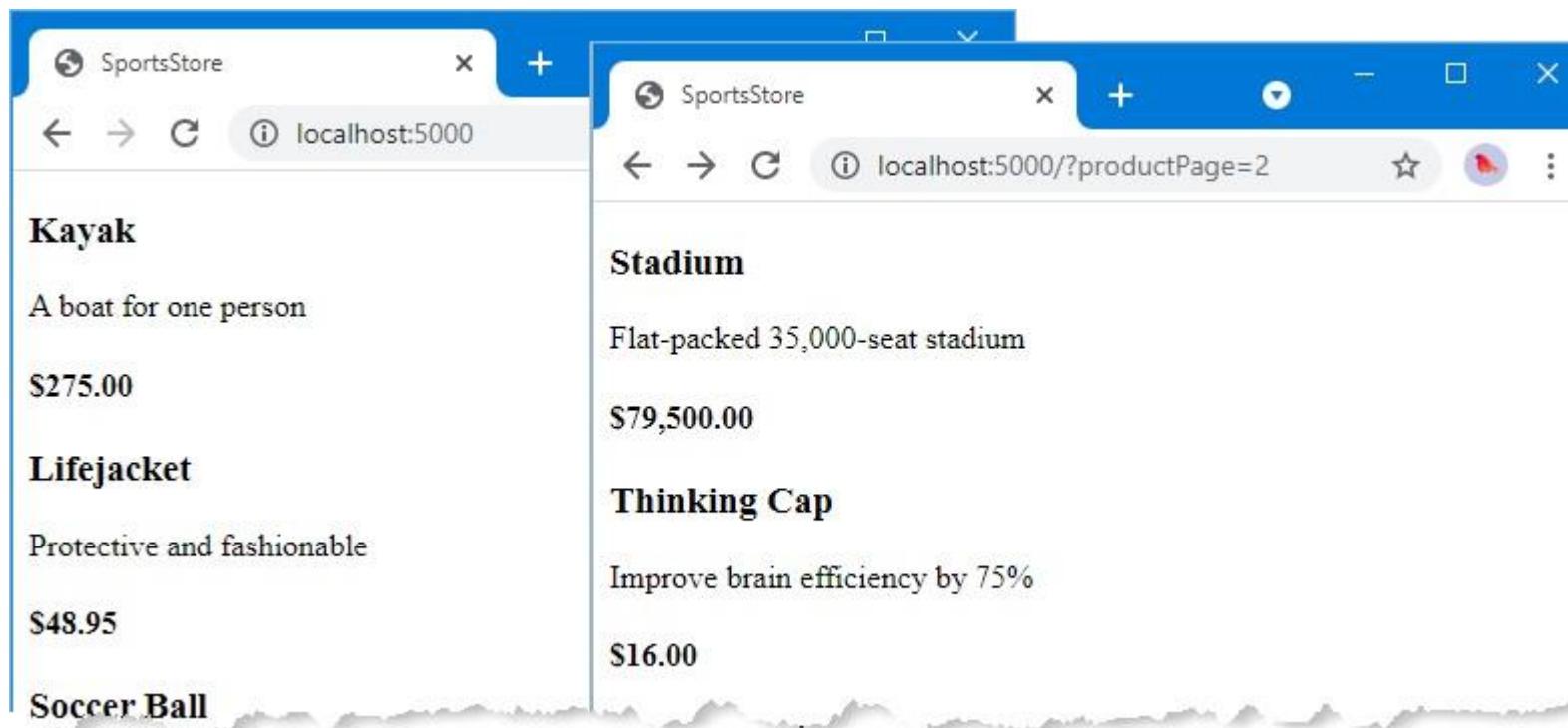
3.3 Running the application

- Request: <http://localhost:5000>



4 Adding pagination

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



³¹ Adding pagination in the 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));
    }
}
```

Controllers/HomeController.cs

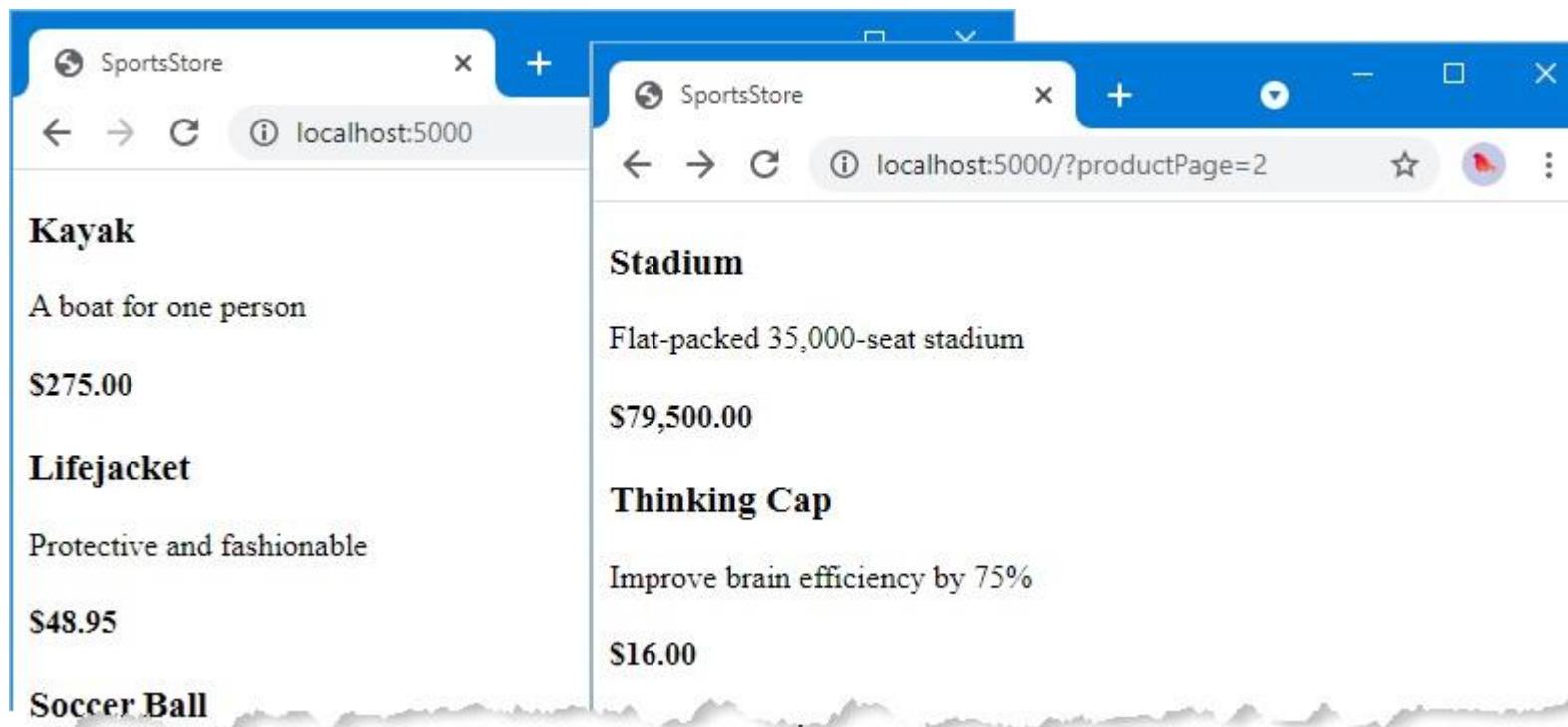
Unit Test: Pagination

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>();
    // 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

```
namespace SportsStore.Models.ViewModels {
```

Models/ViewModels/PagingInfo.cs

```
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

The screenshot shows a Visual Studio interface with two code editors. The left editor displays the implementation of `PageLinkTagHelper.cs`, while the right editor displays its declaration in `Infrastructure/PageLinkTagHelper.cs`. The code editors have dark themes.

```

SportsStore > Infrastructure > C# PageLinkTagHelper.cs > PageLinkTagHelper > .ctor
4   Microsoft.AspNetCore.Mvc.ViewFeatures;
5   Microsoft.AspNetCore.Razor.TagHelpers;
6   SportsStore.Models.ViewModels;
7
8   using SportsStore.Infrastructure;
9
10  [TargetElement("div", Attributes = "page-model")]
11  public class PageLinkTagHelper : TagHelper {
12    private IUrlHelperFactory urlHelperFactory;
13
14    public PageLinkTagHelper(IUrlHelperFactory helperFactory) {
15      urlHelperFactory = helperFactory;
16    }
17
18    [ViewContext]
19    [HtmlAttributeNotBound]
20    public ViewContext? ViewContext { get; set; }
21
22    public PagingInfo? PageModel { get; set; }
23
24    public string? PageAction { get; set; }
25
26    public override void Process(TagHelperContext context,
27      TagHelperOutput output) {
28      if (ViewContext != null && PageModel != null) {
29        IUrlHelper urlHelper
30          = urlHelperFactory.GetUrlHelper(ViewContext);
31        TagBuilder result = new TagBuilder("div");
32        for (int i = 1; i <= PageModel.TotalPages; i++) {
33          TagBuilder tag = new TagBuilder("a");
34          tag.Attributes["href"] = urlHelper.Action(PageAction,
35            new { page = i });
36          tag.InnerHtml.Append(i.ToString());
37          result.InnerHtml.AppendHtml(tag);
38        }
39      }
40    }
41
42

```

Registering a tag helper in the _ViewImports.cshtml file

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

Views/_ViewImports.cshtml

³⁷ UNIT TEST: creating page links

```
File Edit Selection View Go Run ... ⏪ ⏩ SportsSln ... PageLinkTagHelperTests.cs ... PageLinkTagHelperTests.cs ...
```

```
SportsStore.Tests > PageLinkTagHelperTests.cs > ...
```

```
1 using System.Collections.Generic;
2 using System.Threading.Tasks;
3 using Microsoft.AspNetCore.Mvc;
4 using Microsoft.AspNetCore.Mvc.Rendering;
5 using Microsoft.AspNetCore.Mvc.Routing;
6 using Microsoft.AspNetCore.Razor.TagHelpers;
7 using Moq;
8 using SportsStore.Infrastructure;
9 using SportsStore.Models.ViewModels;
10 using Xunit;
11
12 namespace SportsStore.Tests {
13     0 references
14     public class PageLinkTagHelperTests {
15         0 references
16         [Fact]
17         0 references
18         public void Can_Generate_Page_Links() {
19             // Arrange
20             var urlHelper = new Mock<IUrlHelper>();
21             urlHelper.SetupSequence(x =>
22                 x.Action(It.IsAny<UrlActionContext>())
23                     .Returns("Test/Page1")
24                     .Returns("Test/Page2")
25                     .Returns("Test/Page3");
26
27             var urlHelperFactory = new Mock<IUrlHelperFactory>(
28                 urlHelperFactory.Setup(f =>
29                     f.GetUrlHelper(It.IsAny<ActionContext>())
30                         .Returns(urlHelper.Object));
31
32             // Act
33             var result = urlHelperFactory.Object.GetPageLinks(2, 10);
34
35             // Assert
36             Assert.Equal(result.CurrentPage, 2);
37             Assert.Equal(result.TotalItems, 28);
38             Assert.Equal(result.ItemsPerPage, 10);
39             Assert.Equal(result.ViewContext, viewContext.Object);
40             Assert.Equal(result.PageAction, "Test");
41         }
42     }
43 }
44 TagHelperContext ctx = new TagHelperContext(
45     new TagHelperAttributeList(),
46     new Dictionary<object, object>(),
47     "");
48 var content = new Mock<TagHelperContent>();
49 TagHelperOutput output = new TagHelperOutput("div",
50     new TagHelperAttributeList(),
51     (cache, encoder) => Task.FromResult(content.Object));
52
53 // Act
54 helper.Process(ctx, output);
55
56 // Assert
57 Assert.Equal(@"<a href=""Test/Page1"">1</a>" +
58     @"<a href=""Test/Page2"">2</a>" +
59     @"<a href=""Test/Page3"">3</a>",
60     output.Content.GetContent());
61 }
```

Adding the View Model data

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

Models/ViewModels/ProductListViewModel.cs

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

[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);
    Assert.Equal(5, pageInfo.TotalItems);
```

SportsStore.Tests/HomeControllerTests.cs

UNIT TEST: page model view data

```
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);  
}
```

SportsStore.Tests/HomeControllerTests.cs

UNIT TEST: page model view data

[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);
}
```

SportsStore.Tests/HomeControllerTests.cs

Updating the Index.cshtml

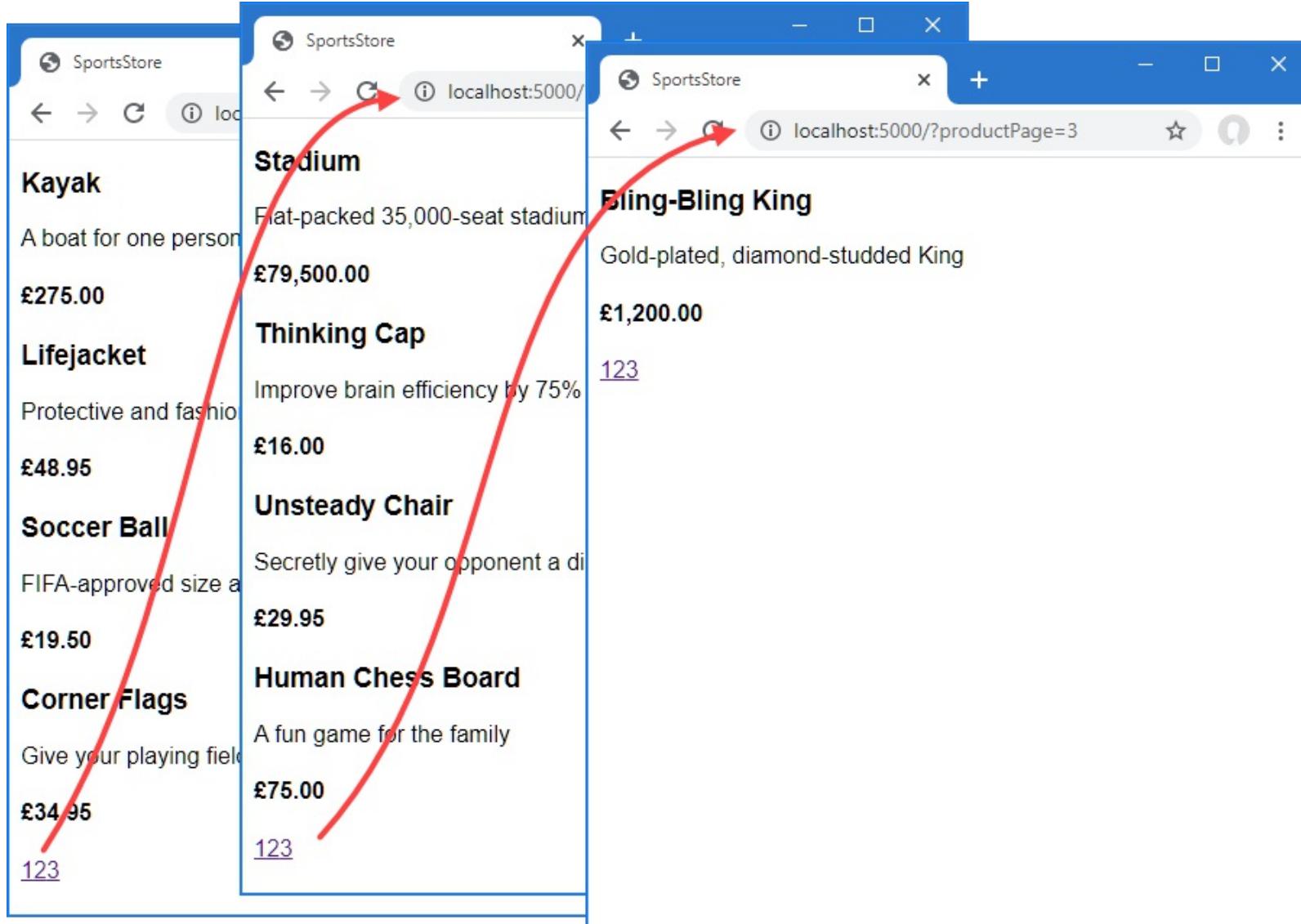
```
@model ProductsListViewModel
```

Views/Home/Index.cshtml

```
@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

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

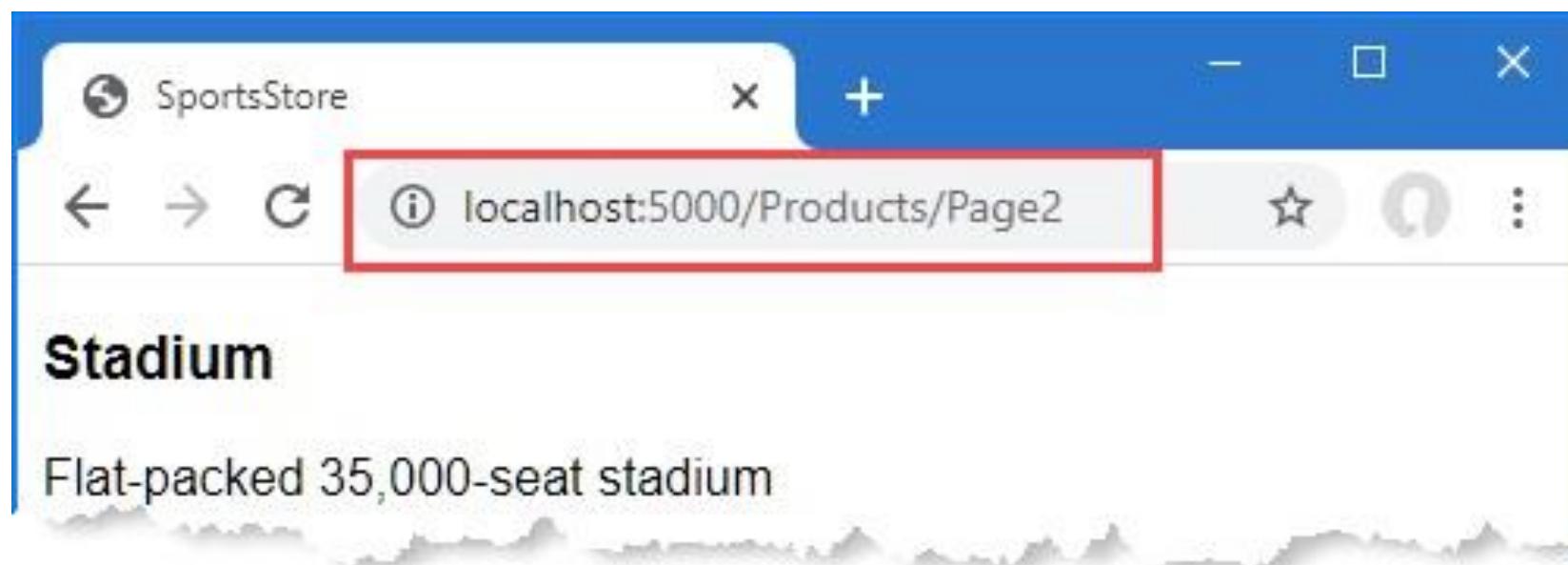
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<StoreDbContext>(opts => {
    opts.UseSqlServer(
        builder.Configuration["ConnectionStrings:SportsStoreConnection"]);
});
builder.Services.AddScoped<IStoreRepository, EFStoreRepository>();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

Program.cs

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

Sports Store (header)

Home

- Watersports
- Soccer
- Chess
- ...

- Product 1
 - Product 2
 - ...
- (main body)

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 Applying Bootstrap styles

```
<!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>
```

Views/Shared/_Layout.cshtml

Styling Content in the 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>
```

Views/Home/Index.cshtml

Adding Classes to Elements in the PageLinkTagHelper.cs

The screenshot shows two instances of the 'PageLinkTagHelper.cs' file in Visual Studio. The left instance is in the 'SportsStore' project, and the right instance is in the 'SportsStore.Infrastructure' project. Both files contain nearly identical code, with the right file showing the result of modifications. Green boxes highlight specific sections of code in both files.

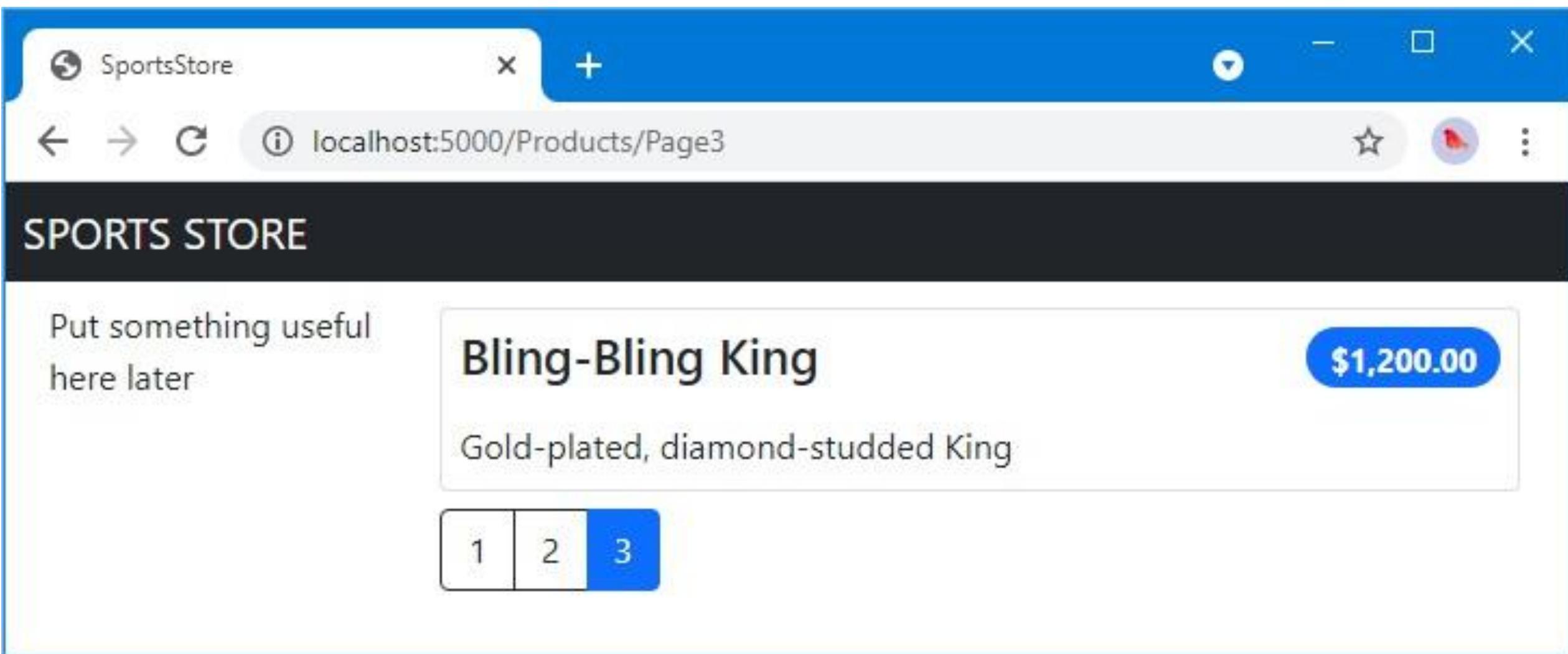
```

SportsStore > Infrastructure > PageLinkTagHelper.cs > PageLinkTagHelper > PageClassSelected
22 public PagingInfo? PageModel { get; set; }
23
24     public string? PageAction { get; set; }
25
26     public bool PageClassesEnabled { get; set; } = false;
27     public string PageClass { get; set; } = String.Empty;
28     public string PageClassNormal { get; set; } = String.Empty;
29     public string PageClassSelected { get; set; } = String.Empty;
30
31     public override void Process(TagHelperContext context,
32         TagHelperOutput output) {
33         if (ViewContext != null && PageModel != null) {
34             IUrlHelper urlHelper
35                 = urlHelperFactory.GetUrlHelper(ViewContext);
36             TagBuilder result = new TagBuilder("div");
37             for (int i = 1; i <= PageModel.TotalPages; i++) {
38                 TagBuilder tag = new TagBuilder("a");
39                 tag.Attributes["href"] = urlHelper.Action(PageAction, new { });
40                 if (PageClassesEnabled) {
41                     tag.AddCssClass(PageClass);
42                     tag.AddCssClass(i == PageModel.CurrentPage
43                         ? PageClassSelected : PageClassNormal);
44                 }
45                 tag.InnerHtml.Append(i.ToString());
46                 result.InnerHtml.AppendHtml(tag);
47             }
48             output.Content.AppendHtml(result.InnerHtml);
49         }
50     }
}

SportsStore > Infrastructure > PageLinkTagHelper.cs > ...
26     public string PageClass { get; set; } = String.Empty;
27     public string PageClassNormal { get; set; } = String.Empty;
28     public string PageClassSelected { get; set; } = String.Empty;
29
30     public override void Process(TagHelperContext context,
31         TagHelperOutput output) {
32         if (ViewContext != null && PageModel != null) {
33             IUrlHelper urlHelper
34                 = urlHelperFactory.GetUrlHelper(ViewContext);
35             TagBuilder result = new TagBuilder("div");
36             for (int i = 1; i <= PageModel.TotalPages; i++) {
37                 TagBuilder tag = new TagBuilder("a");
38                 tag.Attributes["href"] = urlHelper.Action(PageAction, new { });
39                 if (PageClassesEnabled) {
40                     tag.AddCssClass(PageClass);
41                     tag.AddCssClass(i == PageModel.CurrentPage
42                         ? PageClassSelected : PageClassNormal);
43                 }
44                 tag.InnerHtml.Append(i.ToString());
45                 result.InnerHtml.AppendHtml(tag);
46             }
47             output.Content.AppendHtml(result.InnerHtml);
48         }
49     }
}

```

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

```
@model Product
```

Views/Shared/ProductSummary.cshtml

```
<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

```
@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>
```

Views/Index.cshtm

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

