In-class Lab 08

ASP.NET Core MVC

1 Beginning the lab

1. To begin, create a new ASP.NET MVC project by selecting `New` ▶ `Project` ▶ `Visual C#` ▶ `Web` ▶ `.NET Core` ▶ `ASP.NET Core Web Application`. Name the proect `SportsStore` and save the project to your desired location. See figure 1. Click `OK`.

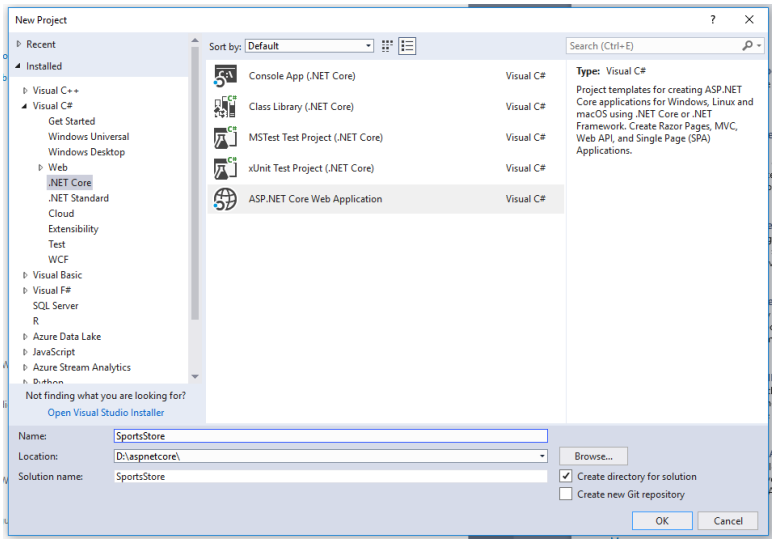

Figure 1: Beginning the tySortsStore application

2. Ensure that `.NET Core` and `ASP.NET Core 2.0` are selectedd in the top tabs and that the `Empty` template is selected. See figure 2. Click `OK`.

3. Create three new folders in the project. Name them `Models`, `Views`, and `Controllers`. See figure 3.

4. Open `Startup.cs` and edit the file as shown in listing 1.

Listing 1: Edit `Startup.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;

namespace SportsStore
```

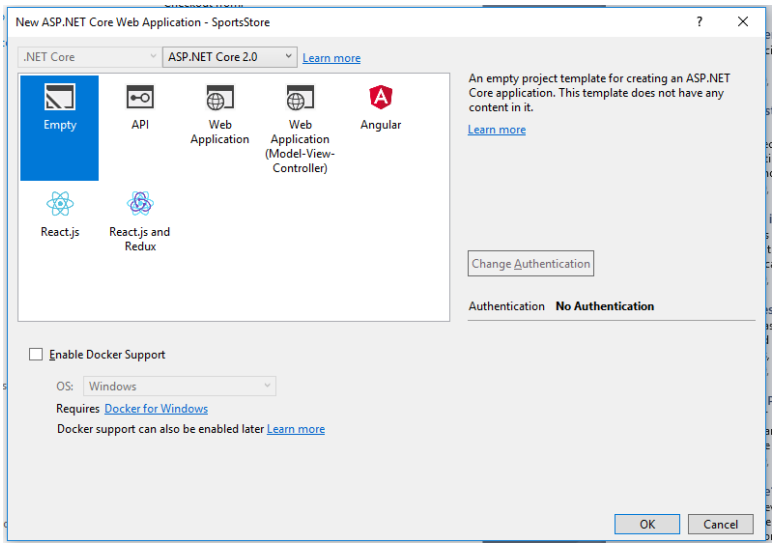

Figure 2: Select the Empty template

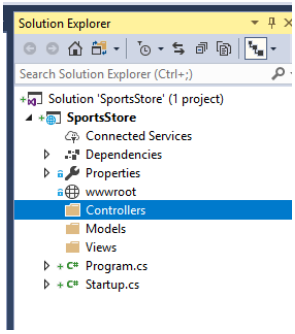

Figure 3: Adding three new folders to project

```
{
    public class Startup
    {
         public void ConfigureServices(IServiceCollection services)
        {
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseDeveloperExceptionPage();
            app.UseStatusCodePages();
            app.UseStaticFiles();
            app.UseMvc(routes =>
                {

                }
            );
        }
```

```
    }
}
```

5. Add a `_ViewImports.cshtml` page to the `Views` folder by right clicking on Views and selecting `Add ► New Item ► Web ► ASP.NET ► Razor View Imports`. See figure 4. Click `Add`.

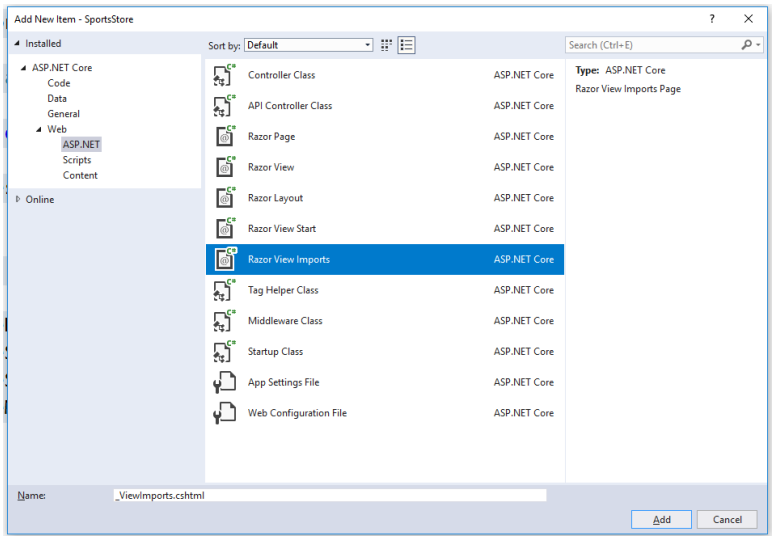

Figure 4: Adding `_ViewImports.cshtml`

6. Edit `_ViewImports` as shown in listing 2.

Listing 2: Editing `_ViewImports.cshtml`

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

7. Add a new test project by right clicking the solution file and selecting `Add ► New Project ► .NET Core ► xUnit Test Project(.NET Core)`. Name the project `SportsStore.Tests`. See figure 5. Click `OK`.

8. Edit the `SportsStore.Tests.csproj` file by right clicking the Tests project and selecting `Edit SportsStore.Tests.csproj` from the pop up menu. Edit the file as shown in listing 8.

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

  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>

    <IsPackable>false</IsPackable>
  </PropertyGroup>

  <ItemGroup>
    <ProjectReference Include="..\SportsStore\SportsStore.csproj" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.NET.Test.Sdk" Version="15.7.0" />
    <PackageReference Include="xunit" Version="2.3.1" />
```

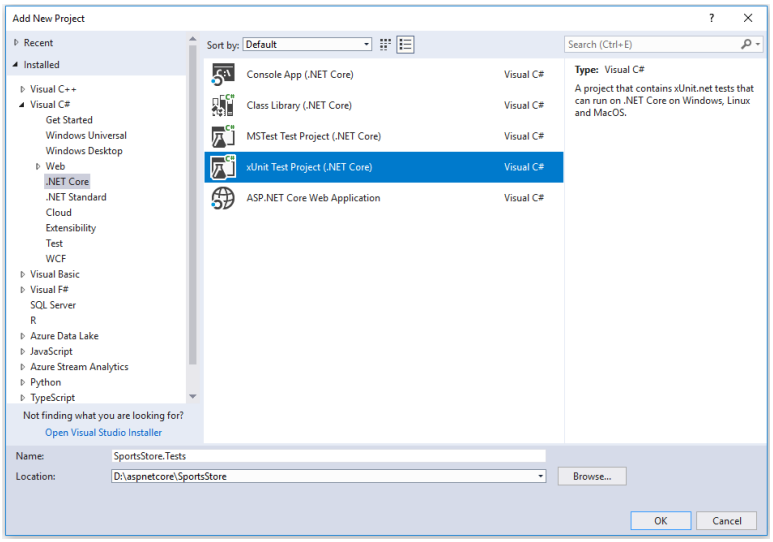

Figure 5: Adding the test project

```
    <PackageReference Include="xunit.runner.visualstudio" Version="2.3.1" />
    <PackageReference Include="Moq" Version="4.7.99" />
  </ItemGroup>

</Project>
```

9. Save all changes to the solution, select `File` ► `Save All`. Check that your Solution Explorer matches figure 6. Build your solution and correct any errors.

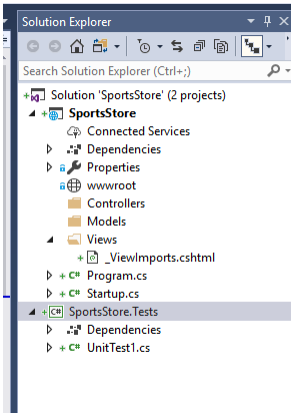

Figure 6: Checking the Solution Explorer

2 Starting the Domain Model

10. Add a class file to the `Models` folder by right clicking `Models` and selecting `Add` ► `Class`. Name the file `Product.cs` and click `Add`.

11. Edit class `Product` as shown in listing 3.

Listing 3: Editing class `Prodduct`

```
namespace SportsStore.Models {

    public class Product {
        public int ProductID { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
        public string Category { get; set; }
    }
}
```

12. Create another class file in `Models` by adding a class in `SportsStore.Models` named `IProductRepository.cs`, and edit the file as shown in listing 4.

Listing 4: Editig class `IProductRepository`

```
using System.Linq;

namespace SportsStore.Models {

    public interface IProductRepository {

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

13. Create a third class file in `Models` by adding a class in `SportsStore.Models` named `FakeProductRepository.cs` and edit the file as shown in listing 13.

```
using System.Collections.Generic;
using System.Linq;

namespace SportsStore.Models {

    public class FakeProductRepository : IProductRepository {

        public IQueryable<Product> Products => new List<Product> {
            new Product { Name = "Football", Price = 25 },
            new Product { Name = "Surf␣board", Price = 179 },
            new Product { Name = "Running␣shoes", Price = 95 }
        }.AsQueryable<Product>();
    }
}
```

14. Revise the `Startup.cs` file by adding two lines, as shown in listing 5.

Listing 5: Revisions to `Startup.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
```

```
using SportsStore.Models;

namespace SportsStore
{
    public class Startup
    {
         public void ConfigureServices(IServiceCollection services)
        {
            services.AddTransient<IProductRepository, FakeProductRepository>();
            services.AddMvc();
        }

        [no change in remainder, omitted from listing]
```

15. Add a controller by right clicking on the `Controllers` file and selecting `Add` ► `Class`. Name the class `ProductController`. Edit the file as shown in listing `lst:asp08h`.

Listing 6: Editing the file `ProductController.cs`

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

namespace SportsStore.Controllers
{
    public class ProductController : Controller
    {
        private IProductRepository repository;

        public ProductController(IProductRepository repo)
        {
            repository = repo;
        }
        public ViewResult List() => View(repository.Products);
    }
}
```

16. Create a new folder under `Views` named `Shared`. The path should be **\Views\Shared**. Right click on the `Views` folder and select tyAdd ► New Folder and name the folder `Shared`.

17. Add a `_Layout.cshtml` file to `\Views\Shared` by right clicking on the `Shared` folder and selecting `Add` ► `Class` ► `ASP.NET Core` ► `Web` ► `Razor Layout`. See figure 7. Click `Add`.

18. Edit the `_Layout.cshtml` file by changing the <title> element as shown in listing 7.

Listing 7: The edit to `_Layout.cshtml`

```
<!DOCTYPE html>

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

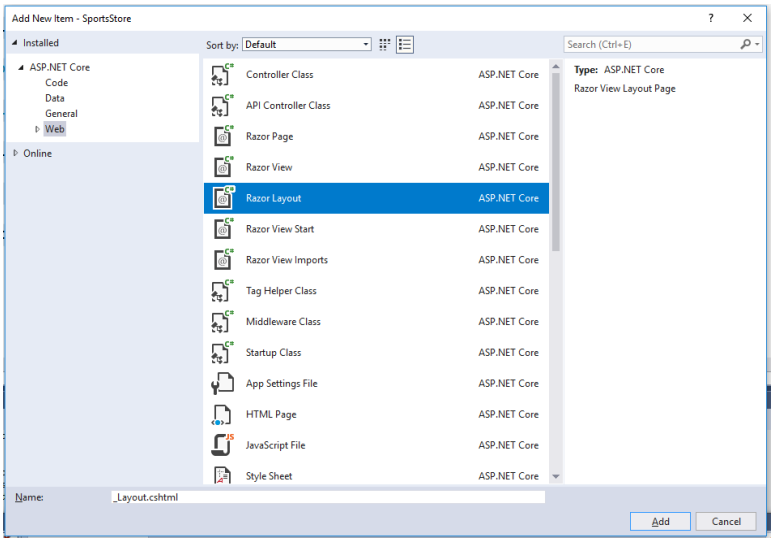

Figure 7: Adding `_Layout.cshtml`

```
</html>
```

19. Add a `_ViewStart.cshtml` file to `Views` by right clicking `Views` and selecting `Add` ► `Class` ► `ASP.NET Core` ► `Web` ► `Razor View Start`. See figure 8. Click `Add`.

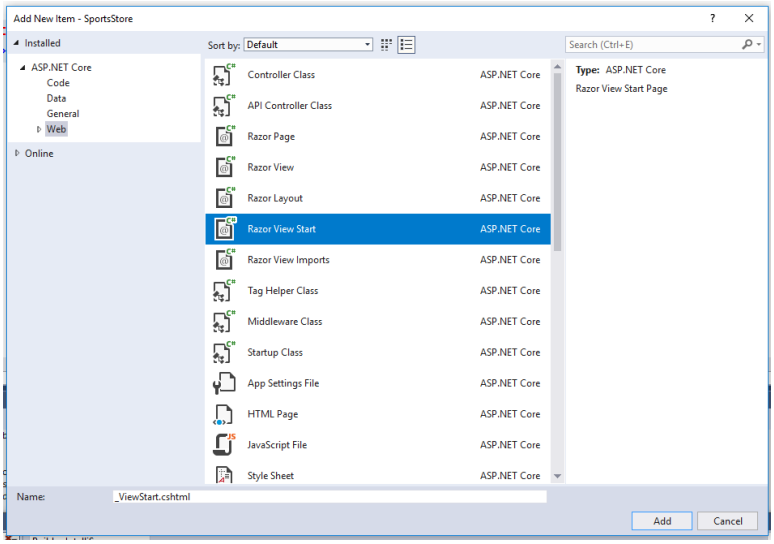

Figure 8: Adding the `_ViewStart.cshtml` file

20. Edit the `_ViewStart.cshtml` file as shown in listing 8.

Listing 8: Editig the `_ViewStart.cshtml` file

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

21. Create another folder under `Views` and name it `Product`.

22. In `\Views\Product` add a Razor View by right clicking the `Product` folder and selecting `Add` ▶ `Class` ▶ `ASP.NET Core` ▶ `Web` ▶ `Razor View` and name it `List.cshtml`. See figure 9. Click `Add`.

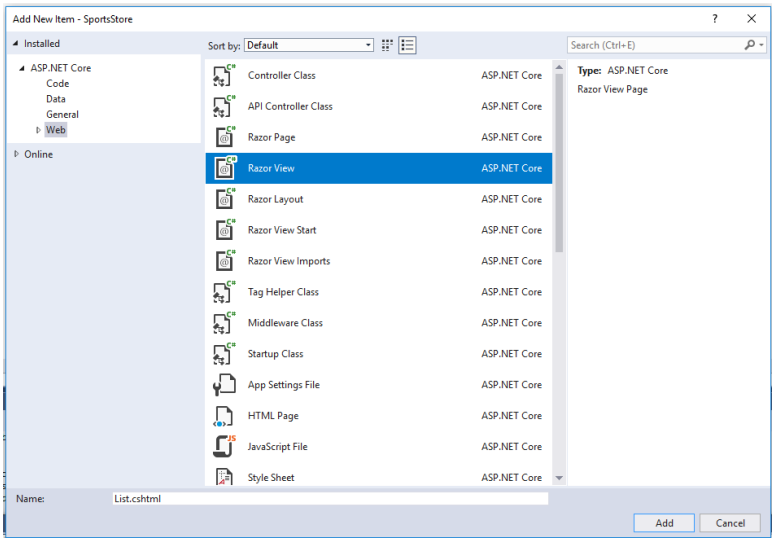

Figure 9: Adding `List.cshtm`

23. Edit `List.cshtml` as shown in listing 9.

Listing 9: Edits to `List.cshtml`

```
@model IEnumerable<Product>

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

24. Edit the `Configure()` method in `Startup.cs` by modifying the call to `apps.MapRoute` as shown in listing 10.

Listing 10: Changing `apps.MapRoute`

```
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseDeveloperExceptionPage();
            app.UseStatusCodePages();
            app.UseStaticFiles();
            app.UseMvc(routes =>
                {
                    routes.MapRoute(name:  "default", template:
                        "controller=Product/action=List/id?");
                }
            );
```

```
        }
```

25. Build the solution and correct any errors. Then, start without debugging.

3 Building the Database

26. Edit the `SportsStore.csproj` file by right clicking on the on the SportsStore project and selecting `Edit SportsStore.csproj`. Edit the file as shown in listing 11.

Listing 11: Editing thhe `SportsStore.csproj` file

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

  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.9" />
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
     Version="2.0.0" />
  </ItemGroup>

</Project>
```

27. Save the solution and build the solution.

28. Add a new class to the `Models` folder. Name it `ApplicationDbContext.cs` Edit the file as shown in listing 28.

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Design;
using Microsoft.Extensions.DependencyInjection;

namespace SportsStore.Models
{

    public class ApplicationDbContext : DbContext
    {

        public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
            : base(options) { }

        public DbSet<Product> Products { get; set; }
    }
}
```

29. Add a new class to the `Models` folder. Name it `EFProductRepository.cs` Edit the file as shown in listing 12.

Listing 12: Editing the `EFProductRepository.cs` file

```
using System.Collections.Generic;
using System.Linq;

namespace SportsStore.Models
{

    public class EFProductRepository : IProductRepository
    {
        private ApplicationDbContext context;

        public EFProductRepository(ApplicationDbContext ctx)
        {
            context = ctx;
        }

        public IQueryable<Product> Products => context.Products;
    }
}
```

30. Add a `appsettings.json` file by right clicking the SportsStore project and selecting `Add` ▶ `New Item` ▶ `General` ▶ `JSON File`. Change the file name to `appsettings.json`. See figure]reffig:asp08j. Click `Add`.

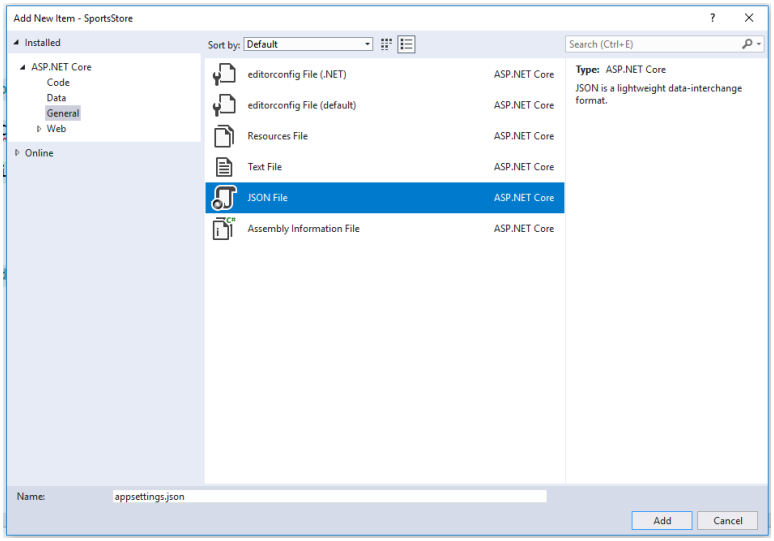

Figure 10: Creating the `appsettings.json` file

31. Edit `appsettings.json` as shown in listing 13. *NOTE: The connection cannot be broken; it must all be on the same line.*

Listing 13: Contents of `appsettings.json`

```
{
  "Data":
  {
    "SportStoreProducts":
    {
```

```
"ConnectionString": "Server=(localdb)\\MSSQLLocalDB;Database=SportsStore;Trusted_Connection=
    True;MultipleActiveResultSets=true"
    }
  }
}
```

32. Edit the `Startup.cs` file as shown in listing 14.

Listing 14: Edits to `Startup.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using SportsStore.Models;
using Microsoft.Extensions.Configuration;
using Microsoft.EntityFrameworkCore;

namespace SportsStore
{
    public class Startup
    {
        public Startup(IConfiguration configuration) =>
            Configuration = configuration;
        public IConfiguration Configuration { get; }
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlServer(
                    Configuration["Data:SportStoreProducts:ConnectionString"]));
            services.AddTransient<IProductRepository, EFProductRepository>();
            services.AddMvc();
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseDeveloperExceptionPage();
            app.UseStatusCodePages();
            app.UseStaticFiles();
            app.UseMvc(routes =>
                {
                    routes.MapRoute(name: "default", template: "{controller=Product}/{action
                        =List}/{id?}");
                }
            );
        }
    }
}
```

33. Make the following change in `Program.cs` in the `BuildWebHost` method, shown in listing 15.

Listing 15: Edit to `Program.cs`

```
public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .UseDefaultServiceProvider(options =>
            options.ValidateScopes = false)
        .Build();
```

34. Open a PowerShell prompt and navigate to the project directory. This will be the one that contains the `Startup.cs` file. Run the following command: `dotnet ef migrations add initial`. See figure 11.

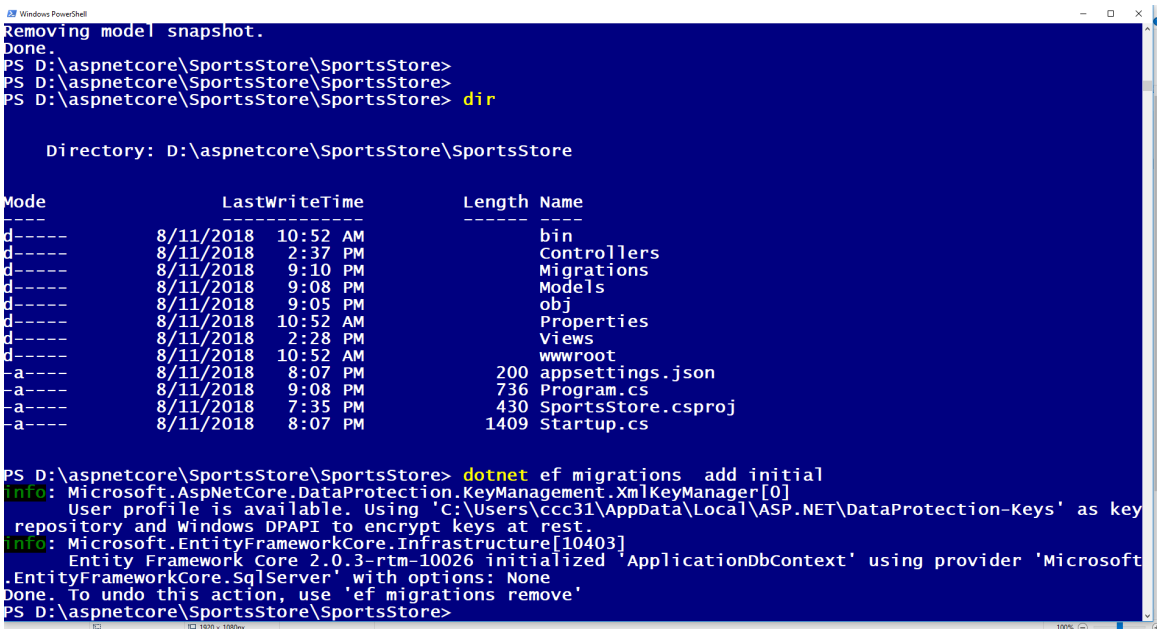

Figure 11: PowerShell session

35. Running the `dotnet ef migrations add initial` will add a `Migrations` folder to your solution. See figure 12.

36. Add a new class named `SeedData.cs` to the `Models` folder and edit it as shown in listing 16.

Listing 16: Contents of `SeedData.cs`

```
using System.Linq;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.EntityFrameworkCore;

namespace SportsStore.Models {

    public static class SeedData {

        public static void EnsurePopulated(IApplicationBuilder app) {
            ApplicationDbContext context = app.ApplicationServices
                .GetRequiredService<ApplicationDbContext>();
            context.Database.Migrate();
            if (!context.Products.Any()) {
```

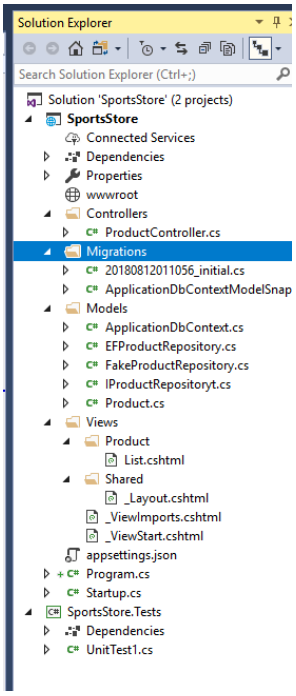

Figure 12: Solution explorer after adding initial migrations

```
            context.Products.AddRange(
                new Product {
                    Name = "Kayak", Description = "A␣boat␣for␣one␣person",
                    Category = "Watersports", Price = 275
                },
                new Product {
                    Name = "Lifejacket",
                    Description = "Protective␣and␣fashionable",
                    Category = "Watersports", Price = 48.95m
                },
                new Product {
                    Name = "Soccer␣Ball",
                    Description = "FIFA-approved␣size␣and␣weight",
                    Category = "Soccer", Price = 19.50m
                },
                new Product {
                    Name = "Corner␣Flags",
                    Description = "Give␣your␣playing␣field␣a␣professional␣touch",
                    Category = "Soccer", Price = 34.95m
                },
                new Product {
                    Name = "Stadium",
                    Description = "Flat-packed␣35,000-seat␣stadium",
                    Category = "Soccer", Price = 79500
                },
                new Product {
                    Name = "Thinking␣Cap",
                    Description = "Improve␣brain␣efficiency␣by␣75%",
                    Category = "Chess", Price = 16
```

```
                    },
                    new Product {
                        Name = "Unsteady Chair",
                        Description = "Secretly give your opponent a disadvantage",
                        Category = "Chess", Price = 29.95m
                    },
                    new Product {
                        Name = "Human Chess Board",
                        Description = "A fun game for the family",
                        Category = "Chess", Price = 75
                    },
                    new Product {
                        Name = "Bling-Bling King",
                        Description = "Gold-plated, diamond-studded King",
                        Category = "Chess", Price = 1200
                    }
                );
                context.SaveChanges();
            }
        }
    }
}
```

37. Add to the `Configure()` method in the `Startup.cs` file a call to the `EnsurePopulated()` method on the `SeedData` object, like this: `SeedData.EnsurePopulated(app);` See listing 17.

Listing 17: Adding `SeedData.EnsurePopulated`

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseDeveloperExceptionPage();
            app.UseStatusCodePages();
            app.UseStaticFiles();
            app.UseMvc(routes =>
                {
                    routes.MapRoute(name: "default", template: "{controller=Product}/{action
                        =List}/{id?}");
                }
            );
            SeedData.EnsurePopulated(app);
        }
```

38. Build your project and correct any errors. Then Start Without Debugging to test it.

4 Pagination

39. Revise the `ProductContrller.cs` file as shown in listing 18.

Listing 18: Revision to `ProductController.cs`

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

namespace SportsStore.Controllers
{
```

```
    public class ProductController : Controller
    {
        private IProductRepository repository;
        public int PageSize = 4;

        public ProductController(IProductRepository repo)
        {
            repository = repo;
        }
        public ViewResult List(int productPage = 1)
                    => View(repository.Products
                            .OrderBy(p => p.ProductID)
                            .Skip((productPage - 1) *PageSize)
                            .Take(PageSize));
    }
}
```

40. Add a file to the `SportsStore.Tests` folder. Name the file `ProductControllerTests.cs`. Edit the file as shown in listing 19.

Listing 19: `ProdductControllerTests.cs` in the `SportsStore.Tests` folder

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

namespace SportsStore.Tests
{
    public class ProductControllerTests
    {
        [Fact]
        public void Can_Paginate()
        {
            // Arrange
            Mock<IProductRepository> mock = new Mock<IProductRepository>();
            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>());

            ProductController controller = new ProductController(mock.Object);
            controller.PageSize = 3;

            // Act
            IEnumerable<Product> result =
                controller.List(2).ViewData.Model as IEnumerable<Product>;

            // Assert
            Product[] prodArray = result.ToArray();
            Assert.True(prodArray.Length == 2);
            Assert.Equal("P4", prodArray[0].Name);
```

```
            Assert.Equal("P5", prodArray[1].Name);
        }
    }
}
```

41. Run the application without debugging. You will have a URI similar to `http://localhost:59231/`. Only four items should display. Thenn change the URI to `http://localhost:59231/?productPage=2`. The next four items should display.

42. In the `SportsStore` project (*not* *`SportsStore.Tests`*) `Models` folder, create a new folder named `ViewModels`. In the new `Models/ViewModels` folder, create a new class named `PagingInfo.cs`. Edit `PagiingInfo.cs` as shown in listing **??**.

Listing 20: the `PagingInfo.cs` class

```
using System;

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);
    }
}
```

43. Create a new folder named `Infrastructure` in the `SportsStore` project (*not* *`SportsStore.Tests`*), and add a new class named `PageLinkTagHelper.cs`. Edit `PageLinkTagHelper.cs` as shown in listing 21.

Listing 21: The `PageLinkTagHelper.cs` class

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Razor.TagHelpers;
using SportsStore.Models.ViewModels;

namespace SportsStore.Infrastructure
{

    [HtmlTargetElement("div", Attributes = "page-model")]
    public class PageLinkTagHelper : TagHelper
    {
        private IUrlHelperFactory urlHelperFactory;
        public PageLinkTagHelper(IUrlHelperFactory helperFactory)
        {
            urlHelperFactory = helperFactory;
        }
        [ViewContext]
        [HtmlAttributeNotBound]
```

```
        public ViewContext ViewContext { get; set; }

        public PagingInfo PageModel { get; set; }

        public string PageAction { get; set; }

        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            IUrlHelper urlHelper = urlHelperFactory.GetUrlHelper(ViewContext);
            TagBuilder result = new TagBuilder("div");
            for (int i = 1; i <= PageModel.TotalPages; i++)
            {
                TagBuilder tag = new TagBuilder("a");
                tag.Attributes["href"] = urlHelper.Action(PageAction,
                   new { productPage = i });
                tag.InnerHtml.Append(i.ToString());
                result.InnerHtml.AppendHtml(tag);
            }
            output.Content.AppendHtml(result.InnerHtml);
        }
    }
}
```

44. Edit the `_ViewImports.cshtml` file by adding the two lines shown in listing 22.

Listing 22: Revising the `_ViewImports.cshtml` file

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

45. Create a new test in `SportsStore.Tests` named `PageLinkTagHelperTests.cs`. Edit the file as shown in 23. Then, Run All tests.

Listing 23: `PageLinkTagHelperTests.cs`

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

namespace SportsStore.Tests
{

    public class PageLinkTagHelperTests
    {

        [Fact]
        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>()))
                        .Returns(urlHelper.Object);

            PageLinkTagHelper helper =
                    new PageLinkTagHelper(urlHelperFactory.Object)
                    {
                        PageModel = new PagingInfo
                        {
                            CurrentPage = 2,
                            TotalItems = 28,
                            ItemsPerPage = 10
                        },
                        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.Object));

            // 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());
        }
    }
}
```

46. Create a new class file in `Models/ViewModels` named `ProductsListViewModel.cs` and edit it as shown in listing 24.

Listing 24: `Models/ViewModels/ProductsListViewModel.cs`

```
using System;
using System.Collections.Generic;
using SportsStore.Models;

namespace SportsStore.Models.ViewModels
{
    public class ProductsListViewModel
    {
```

```
        public IEnumerable<Product> Products { get; set; }
        public PagingInfo PagingInfo { get; set; }
    }
}
```

47. Edit the `ProductCotroller.cs` file by making the changes shown in listing 25.

Listing 25: Revisions to `ProductController.cs`

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

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

        public ProductController(IProductRepository repo)
        {
            repository = repo;
        }
        public ViewResult List(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()
                        }
                    });
    }
}
```

48. Revise `ProductControllerTests.cs` as shown in listing ??. There a number of changes so please carefully compare the files.

```
using System.Collections.Generic;
using System.Linq;
using Moq;
using SportsStore.Controllers;
using SportsStore.Models;
using Xunit;
using SportsStore.Models.ViewModels;

namespace SportsStore.Tests
{

    public class ProductControllerTests
```

```
{

    [Fact]
    public void Can_Paginate()
    {
        // Arrange
        Mock<IProductRepository> mock = new Mock<IProductRepository>();
        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>());

        ProductController controller = new ProductController(mock.Object);
        controller.PageSize = 3;

        // Act
        ProductsListViewModel result =
            controller.List(2).ViewData.Model as ProductsListViewModel;

        // Assert
        Product[] prodArray = result.Products.ToArray();
        Assert.True(prodArray.Length == 2);
        Assert.Equal("P4", prodArray[0].Name);
        Assert.Equal("P5", prodArray[1].Name);
    }


    [Fact]
    public void Can_Send_Pagination_View_Model()
    {

        // Arrange
        Mock<IProductRepository> mock = new Mock<IProductRepository>();
        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
        ProductController controller =
            new ProductController(mock.Object) { PageSize = 3 };

        // Act
        ProductsListViewModel result =
            controller.List(2).ViewData.Model as ProductsListViewModel;

        // Assert
        PagingInfo pageInfo = result.PagingInfo;
        Assert.Equal(2, pageInfo.CurrentPage);
        Assert.Equal(3, pageInfo.ItemsPerPage);
```

```
            Assert.Equal(5, pageInfo.TotalItems);
            Assert.Equal(2, pageInfo.TotalPages);
        }

    }
}
```

49. Run All tests to make sure everything is working as it should.

50. Update the `List.cshtml` file as shown in listing 26.

Listing 26: Updates to `List.cshtml`

```
@model ProductsListViewModel

@foreach (var p in Model.Products)
{
<div>
    <h3>@p.Name</h3>
    @p.Description
    <h4>@p.Price.ToString("c")</h4>
</div>
}

<div page-model="@Model.PagingInfo" page-action="List"></div></div>
```

51. Revise the `Configure` method in the `Startup.cs` file to add a "paginatiob" route as shown in listing 27.

Listing 27: Edit to `Confiigure()`

```
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseDeveloperExceptionPage();
            app.UseStatusCodePages();
            app.UseStaticFiles();
            app.UseMvc(routes => {
                routes.MapRoute(
                    name:  "pagination",
                    template:  "Products/PageproductPage",
                    defaults:  new  Controller = "Product", action = "List" );
                routes.MapRoute(
                    name: "default",
                    template: "{controller=Product}/{action=List}/{id?}");
            });
```

52. Start your application without debugging and try out the page links.

5 Improving the User Experience

53. Edit the `_Layout.cshtml` in the `Views/Shared` folder as shown in listing 28.

Listing 28: Edit to _Layout.cshtml

```
<!DOCTYPE html>
```

```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/css/
        bootstrap.min.css" integrity="sha384-WskhaSGFgHYWDcbwN70/dfYBj47jz9qbsMId/
        iRN3ewGhXQFZCSftd1LZCfmhktB" crossorigin="anonymous">
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.1/js/bootstrap.min.js"
        integrity="sha384-smHYKdLADwkXOn1EmN1qk/HfnUcbVRZyYmZ4qpPea6sjB/pTJ0euyQp0Mk8ck+5T"
        crossorigin="anonymous"></script>
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X
        +965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"
        ></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.3/umd/popper.min.js"
        integrity="sha384-ZMP7rVo3mIykV+2+9J3UJ46jBk0WLaUAdn689aCwoqbBJiSnjAK/l8WvCWPIPm49"
        crossorigin="anonymous"></script>
    <title>SportsStore</title>
</head>
<body>
    <div class="navbar navbar-inverse bg-inverse" role="navigation">
        <a class="navbar-brand" href="#">SPORTS STORE</a>
    </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>
```

54. Edit `List.cshtml` in the `Views/Product` folder as shown in listing 29.

Listing 29: Edit to `List.cshtml`

```
@model ProductsListViewModel

@foreach (var p in Model.Products)
{
<div class="card card-outline-primary m-1 p-1">
    <div class="bg-faded p-1">
        <h4>@p.Name
            <span class="badge badge-pill badge-primary" 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="List" page-classes-enabled="true"
     page-class="btn" page-class-normal="btn-secondary"
     page-class-selected="btn-primary" class="btn-group pull-right m-1">
```

```
</div>
```

55. Edit the `PageLinkTagHelpers.cs` file as shown in listing 30.

Listing 30: Edit to `PageLinkTagHelpers.cs`

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Mvc.ViewFeatures;
using Microsoft.AspNetCore.Razor.TagHelpers;
using SportsStore.Models.ViewModels;

namespace SportsStore.Infrastructure
{

    [HtmlTargetElement("div", Attributes = "page-model")]
    public class PageLinkTagHelper : TagHelper
    {
        private IUrlHelperFactory urlHelperFactory;
        public PageLinkTagHelper(IUrlHelperFactory helperFactory)
        {
            urlHelperFactory = helperFactory;
        }
        [ViewContext]
        [HtmlAttributeNotBound]
        public ViewContext ViewContext { get; set; }

        public PagingInfo PageModel { get; set; }

        public string PageAction { get; set; }
        public bool PageClassesEnabled { get; set; } = false;
        public string PageClass { get; set; }
        public string PageClassNormal { get; set; }
        public string PageClassSelected { get; set; }

        public override void Process(TagHelperContext context, TagHelperOutput output)
        {
            IUrlHelper urlHelper = urlHelperFactory.GetUrlHelper(ViewContext);
            TagBuilder result = new TagBuilder("div");
            for (int i = 1; i <= PageModel.TotalPages; i++)
            {
                TagBuilder tag = new TagBuilder("a");
                tag.Attributes["href"] = urlHelper.Action(PageAction,
                   new { productPage = i });
                if (PageClassesEnabled)
                {
                    tag.AddCssClass(PageClass);
                    tag.AddCssClass(i == PageModel.CurrentPage
                        ? PageClassSelected : PageClassNormal);
                }
                tag.InnerHtml.Append(i.ToString());
                result.InnerHtml.AppendHtml(tag);
            }
            output.Content.AppendHtml(result.InnerHtml);
        }
    }
```

```
}
```

56. Create a new file named `ProductSummary.cshtml` in the `Views/Shared` folder and eidt it as shown in listing 31.

Listing 31: Create a new file named `ProductSummary.cshtnml`

```
@model Product

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

57. Edit `List.cshtml` as shown in listing 32 to use the partial page view.

Listing 32: Using a partiall page in `List.cshtml`

```
@model ProductsListViewModel

@foreach (var p in Model.Products) {
    @Html.Partial("ProductSummary", p)
}

<div page-model="@Model.PagingInfo" page-action="List" page-classes-enabled="true"
     page-class="btn" page-class-normal="btn-secondary"
     page-class-selected="btn-primary" class="btn-group␣pull-right␣m-1">
</div>
```