In-class Lab 09

ASP.NET Core MVC

1 Beginning the lab

1. This lab builds on the lab in chapter 8, `SportsStore`. To begin with a clean code base, copy the entire folder from your chapter 8 project into a new folder named `SportsStoreChapter09`. This represents the completed code for chapter 8. Build the project and then Start Without Debugging.

   Database error If you receive a database error, you will need to comment out the following two lines. In `Startup.cs`, comment out:

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

   In `SeedData.cs`, comment out the following line:

   ```
   context.Database.Migrate();
   ```

   The problem is that you are using the same database you created for chapter 8, so it alredy exists. Commenting out these two lines will prevent Entity Framework from attempting to build an existing database. After this, you should be able to Start Wthout Debugging and achieve the same results as in the last lab.

2. Open `Models/ViewModels/ProductsListViewModel.cs` and add the line shown in listing 1.

   Listing 1: Edit to `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; }
           public string CurrentCategory  get; set;
       }
   }
   ```

3. Edit `Controllers/ProductController.cs` by make the changes shown in listing 2.

   Listing 2: Additions to `ProductCobtroller.cs`

   ```
       public ViewResult List(string category, int productPage = 1)
               => View(new ProductsListViewModel
               {
   ```

```
            Products = repository.Products
                .Where(p => category == null || p.Category == category)
                .OrderBy(p => p.ProductID)
                .Skip((productPage - 1) * PageSize)
                .Take(PageSize),
            PagingInfo = new PagingInfo
            {
                CurrentPage = productPage,
                ItemsPerPage = PageSize,
                TotalItems = repository.Products.Count()
            },
            CurrentCategory = category
        });
```

4. Edit the `Can_Paginate()` method in `SportsStore.Tests/ProductControllerTests.cs` as shown in listiing 3.

Listing 3: Edit `Can_Paginate()` method

```
    [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(null, 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);
    }
```

5. Make the same change to the `Can_Sent_Pagination_View_Model()` in `SportsStore.Tests/ProductContro`
as shown in listing 4.

Listing 4: Edit

```
    [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(null, 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);
    }
```

6. Run All Tests. Then Build the solution and Start Without Debugging. Then, navigate to `http://localhost:59231/?category=Soccer` (taking care to change the port on `localhost`).

7. Add a `Can_Filter_Products()` method to `ProductControllerTests.cs` as shown in listing 5. Then, Run All Tests.

Listing 5: Adding method `Can_Filter_Products()`

```
    [Fact]
    public void Can\_Filter\_Products()
    {
        // Arrange
        // - create the mock repository
        Mock<IProductRepository> mock = new Mock<IProductRepository>();
        mock.Setup(m => m.Products).Returns((new Product[] {
            new Product {ProductID = 1, Name = "P1", Category = "Cat1"},
            new Product {ProductID = 2, Name = "P2", Category = "Cat2"},
            new Product {ProductID = 3, Name = "P3", Category = "Cat1"},
            new Product {ProductID = 4, Name = "P4", Category = "Cat2"},
            new Product {ProductID = 5, Name = "P5", Category = "Cat3"}
        }).AsQueryable<Product>());

        // Arrange - create a controller and make the page size 3 items
        ProductController controller = new ProductController(mock.Object);
        controller.PageSize = 3;

        // Action
        Product[] result =
            (controller.List("Cat2", 1).ViewData.Model as ProductsListViewModel)
                .Products.ToArray();

        // Assert
        Assert.Equal(2, result.Length);
        Assert.True(result[0].Name == "P2" && result[0].Category == "Cat2");
```

```
            Assert.True(result[1].Name == "P4" && result[1].Category == "Cat2");
        }
```

2 Changing the Routing Scheme

8. Edit the `Configure()` method in `Startup.cs` as shown in listing 6.

Listing 6: Changing the routig in `Startup.cs`

```
app.UseMvc(routes => {
        routes.MapRoute(
            name: null,
            template: "{category}/Page{productPage:int}",
            defaults: new { controller = "Product", action = "List" }
        );

        routes.MapRoute(
            name: null,
            template: "Page{productPage:int}",
            defaults: new { controller = "Product", action = "List", productPage = 1 }
        );

        routes.MapRoute(
            name: null,
            template: "{category}",
            defaults: new { controller = "Product", action = "List", productPage = 1 }
        );

        routes.MapRoute(
            name: null,
            template: "",
            defaults: new { controller = "Product", action = "List", productPage = 1 });

        routes.MapRoute(name: null, template: "{controller}/{action}/{id?}");
    });
```

9. Make the following changes to the `PageLinkTagHelpers.cs` file.

- Add the following line to the `using` directives at the top of the file:

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

- Add a `page-url` attribute as follows:

```
        [HtmlAttributeName(DictionaryAttributePrefix = "page-url-")]
        public Dictionary<string, object> PageUrlValues  get; set;
            = new Dictionary<string, object>();
```

- Add the following lines to the `Process()` method:

```
        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");
                PageUrlValues["productPage"] = i;
                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);
        }
```

10. Edit `List.cshtml` as shown in listing 7.

Listing 7: Changes to `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" page-url-category="@Model.CurrentCategory"
     class="btn-group␣pull-right␣m-1">
</div>
```

11. Create a new folder in the `SportsStore` project named `Components`. Inside that folder, create a new class fie named `NavigationMenuViewComponent.cs`. Edit the file so that it matches listing 8.

Listing 8: `NavigationMenuViewComponentcs`

```
using Microsoft.AspNetCore.Mvc;

namespace SportsStore.Components
{

    public class NavigationMenuViewComponent : ViewComponent
    {
        public string Invoke()
        {
            return "Hello␣from␣NavigationMenuViewComponent.";
        }
    }
}
```

12. Edit `_Layout.cshtml` to match listing 9. Then, Start Without Debugging.

Listing 9: Edit to `_Layout.cshtl`

```
<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->
            @await Component.InvokeAsync("NavigationMenu")
        </div>
        <div class="col-9">
            @RenderBody()
        </div>
    </div>
</body>
```

13. Edit `NavigationMenuViewComponent.cs` as sown in listing 10.

Listing 10: Edits to `NavigationMenuVieComponent.cs`

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

namespace SportsStore.Components {

    public class NavigationMenuViewComponent : ViewComponent {
        private IProductRepository repository;

        public NavigationMenuViewComponent(IProductRepository repo) {
            repository = repo;
        }

        public IViewComponentResult Invoke() {
            return View(repository.Products
                .Select(x => x.Category)
                .Distinct()
                .OrderBy(x => x));
        }
    }
}
```

14. Create a new clas file in `SportsStore.Tests` named `NavigatiobMenuViewConponentTests.cs`, and edit the file to match listing 11.

Listing 11: New class file named `NavigatiobMenuViewCoponentTests.cs`

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

namespace SportsStore.Tests
{
    public class NavigationMenuViewComponentTests
    {
```

```
        [Fact]
        public void Can_Select_Categories()
        {
            // Arrange
            Mock<IProductRepository> mock = new Mock<IProductRepository>();
            mock.Setup(m => m.Products).Returns((new Product[] {
                new Product {ProductID = 1, Name = "P1", Category = "Apples"},
                new Product {ProductID = 2, Name = "P2", Category = "Apples"},
                new Product {ProductID = 3, Name = "P3", Category = "Plums"},
                new Product {ProductID = 4, Name = "P4", Category = "Oranges"},
            }).AsQueryable<Product>());

            NavigationMenuViewComponent target =
                new NavigationMenuViewComponent(mock.Object);

            // Act = get the set of categories
            string[] results = ((IEnumerable<string>)(target.Invoke()
                as ViewViewComponentResult).ViewData.Model).ToArray();

            // Assert
            Assert.True(Enumerable.SequenceEqual(new string[] { "Apples",
                "Oranges", "Plums" }, results));
        }
    }
}
```

15. Create a new folder with the path `Views/Shared/Components/NavigationMenu`. Create a new view file named `Default.cshtml`. Place the new file in `Views/Shared/Components/NavigationMenu` folder. Edit the file to match listing 12.

Listing 12: Cotents of `Default.cshtml`

```
@model IEnumerable<string>

<a class="btn␣btn-block␣btn-secondary"
   asp-action="List"
   asp-controller="Product"
   asp-route-category="">
    Home
</a>

@foreach (string category in Model)
{
    <a class="btn␣btn-block
␣␣␣␣␣␣␣@(category␣==␣ViewBag.SelectedCategory␣?␣"btn-primary":␣"btn-secondary")"
       asp-action="List"
       asp-controller="Product"
       asp-route-category="@category"
       asp-route-productPage="1">
        @category
    </a>
}
```

16. Revise the `NavigationMenuViewComponent.cs` file by adding the line shown below in the `Invoke()` method. See listing 13.

Listing 13: Revision to `Invoke()` in `NavigationMenuViewCommponent.cs`

```
public IViewComponentResult Invoke()
{
    ViewBag.SelectedCategory = RouteData?.Values["category"];
    return View(repository.Products
        .Select(x => x.Category)
        .Distinct()
        .OrderBy(x => x));
}
```

17. Add two `using` statements to `NavigationMenuViewComponentTests.cs`. These are shown below. Add a new test to `NavigationMenuViewComponentTests.cs` named `Indicates_Selected_Category()`. This test is shown in listing 14.

Listing 14: Test `Indicates_Selected_Category()`

```
using Microsoft.AspNetCore.Mvc.Rendering;
using Microsoft.AspNetCore.Routing;

public void Indicates_Selected_Category()
{

    // Arrange
    string categoryToSelect = "Apples";
    Mock<IProductRepository> mock = new Mock<IProductRepository>();
    mock.Setup(m => m.Products).Returns((new Product[] {
        new Product {ProductID = 1, Name = "P1", Category = "Apples"},
        new Product {ProductID = 4, Name = "P2", Category = "Oranges"},
    }).AsQueryable<Product>());
    NavigationMenuViewComponent target =
        new NavigationMenuViewComponent(mock.Object);
    target.ViewComponentContext = new ViewComponentContext
    {
        ViewContext = new ViewContext
        {
            RouteData = new RouteData()
        }
    };
    target.RouteData.Values["category"] = categoryToSelect;

    // Action
    string result = (string)(target.Invoke() as
        ViewViewComponentResult).ViewData["SelectedCategory"];

    // Assert
    Assert.Equal(categoryToSelect, result);
}
```

18. Revise the `List()` method in the file `ProductController.cs` file as shown in listing 15.

Listing 15: Revision to `List()` method in `ProductController.cs`

```
public ViewResult List(string category, int productPage = 1)
            => View(new ProductsListViewModel
            {
                Products = repository.Products
                    .Where(p => category == null || p.Category == category)
```

```
                        .OrderBy(p => p.ProductID)
                        .Skip((productPage - 1) * PageSize)
                        .Take(PageSize),
                    PagingInfo = new PagingInfo
                    {
                        CurrentPage = productPage,
                        ItemsPerPage = PageSize,
                        TotalItems = category == null ?
                            repository.Products.Count() :
                            repository.Products.Where(e =>
                                e.Category == category).Count()
                    },
                    CurrentCategory = category
                });
```

19. Add a new test named `Generat_Category_Specific_Product_Count()` in `ProductCotrollerTests.cs` as shown in listing 19. Run All Tests.

```
using System;
using Microsoft.AspNetCore.Mvc;

        [Fact]
        public void Generate\_Category\_Specific\_Product\_Count()
        {
            // Arrange
            Mock<IProductRepository> mock = new Mock<IProductRepository>();
            mock.Setup(m => m.Products).Returns((new Product[] {
                new Product {ProductID = 1, Name = "P1", Category = "Cat1"},
                new Product {ProductID = 2, Name = "P2", Category = "Cat2"},
                new Product {ProductID = 3, Name = "P3", Category = "Cat1"},
                new Product {ProductID = 4, Name = "P4", Category = "Cat2"},
                new Product {ProductID = 5, Name = "P5", Category = "Cat3"}
            }).AsQueryable<Product>());

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

            Func<ViewResult, ProductsListViewModel> GetModel = result =>
                result?.ViewData?.Model as ProductsListViewModel;

            // Action
            int? res1 = GetModel(target.List("Cat1"))?.PagingInfo.TotalItems;
            int? res2 = GetModel(target.List("Cat2"))?.PagingInfo.TotalItems;
            int? res3 = GetModel(target.List("Cat3"))?.PagingInfo.TotalItems;
            int? resAll = GetModel(target.List(null))?.PagingInfo.TotalItems;

            // Assert
            Assert.Equal(2, res1);
            Assert.Equal(2, res2);
            Assert.Equal(1, res3);
            Assert.Equal(5, resAll);
        }
```

3 Adding a Shopping Cart

20. Add a new class file named `Cart.cs` to the `Models` folder. Edit the file as shown in listing 16.

Listing 16: `Cart.cs`

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

namespace SportsStore.Models
{
    public class Cart
    {
        private List<CartLine> lineCollection = new List<CartLine>();

        public virtual void AddItem(Product product, int quantity)
        {
            CartLine line = lineCollection
                .Where(p => p.Product.ProductID == product.ProductID)
                .FirstOrDefault();

            if (line == null)
            {
                lineCollection.Add(new CartLine
                {
                    Product = product,
                    Quantity = quantity
                });
            }
            else
            {
                line.Quantity += quantity;
            }
        }

        public virtual void RemoveLine(Product product) =>
            lineCollection.RemoveAll(l => l.Product.ProductID == product.ProductID);

        public virtual decimal ComputeTotalValue() =>
            lineCollection.Sum(e => e.Product.Price * e.Quantity);

        public virtual void Clear() => lineCollection.Clear();

        public virtual IEnumerable<CartLine> Lines => lineCollection;
    }

    public class CartLine
    {
        public int CartLineID { get; set; }
        public Product Product { get; set; }
        public int Quantity { get; set; }
    }
}
```

21. Add a new test file named `CartTests.cs` to the `SportsStore.Tests` project. Edit the file as shown in listing 17.

Listing 17: CartTests.cs

```
using System.Linq;
using SportsStore.Models;
using Xunit;

namespace SportsStore.Tests
{
    public class CartTests
    {
        [Fact]
        public void Can\_Add\_New\_Lines()
        {
            // Arrange - create some test products
            Product p1 = new Product { ProductID = 1, Name = "P1" };
            Product p2 = new Product { ProductID = 2, Name = "P2" };

            // Arrange - create a new cart
            Cart target = new Cart();

            // Act
            target.AddItem(p1, 1);
            target.AddItem(p2, 1);
            CartLine[] results = target.Lines.ToArray();

            // Assert
            Assert.Equal(2, results.Length);
            Assert.Equal(p1, results[0].Product);
            Assert.Equal(p2, results[1].Product);
        }

        [Fact]
        public void Can\_Add\_Quantity\_For\_Existing\_Lines()
        {
            // Arrange - create some test products
            Product p1 = new Product { ProductID = 1, Name = "P1" };
            Product p2 = new Product { ProductID = 2, Name = "P2" };

            // Arrange - create a new cart
            Cart target = new Cart();

            // Act
            target.AddItem(p1, 1);
            target.AddItem(p2, 1);
            target.AddItem(p1, 10);
            CartLine[] results = target.Lines
                .OrderBy(c => c.Product.ProductID).ToArray();

            // Assert
            Assert.Equal(2, results.Length);
            Assert.Equal(11, results[0].Quantity);
            Assert.Equal(1, results[1].Quantity);
        }

        [Fact]
        public void Can\_Remove\_Line()
        {
```

```
            // Arrange - create some test products
            Product p1 = new Product { ProductID = 1, Name = "P1" };
            Product p2 = new Product { ProductID = 2, Name = "P2" };
            Product p3 = new Product { ProductID = 3, Name = "P3" };

            // Arrange - create a new cart
            Cart target = new Cart();
            // Arrange - add some products to the cart
            target.AddItem(p1, 1);
            target.AddItem(p2, 3);
            target.AddItem(p3, 5);
            target.AddItem(p2, 1);

            // Act
            target.RemoveLine(p2);

            // Assert
            Assert.Equal(0, target.Lines.Where(c => c.Product == p2).Count());
            Assert.Equal(2, target.Lines.Count());
        }

        [Fact]
        public void Calculate\_Cart\_Total()
        {
            // Arrange - create some test products
            Product p1 = new Product { ProductID = 1, Name = "P1", Price = 100M };
            Product p2 = new Product { ProductID = 2, Name = "P2", Price = 50M };

            // Arrange - create a new cart
            Cart target = new Cart();

            // Act
            target.AddItem(p1, 1);
            target.AddItem(p2, 1);
            target.AddItem(p1, 3);
            decimal result = target.ComputeTotalValue();

            // Assert
            Assert.Equal(450M, result);
        }

        [Fact]
        public void Can\_Clear\_Contents()
        {
            // Arrange - create some test products
            Product p1 = new Product { ProductID = 1, Name = "P1", Price = 100M };
            Product p2 = new Product { ProductID = 2, Name = "P2", Price = 50M };

            // Arrange - create a new cart
            Cart target = new Cart();

            // Arrange - add some items
            target.AddItem(p1, 1);
            target.AddItem(p2, 1);

            // Act - reset the cart
```

```
            target.Clear();

            // Assert
            Assert.Equal(0, target.Lines.Count());
        }
    }
}
```

22. Add a class file namd `UrlExtensions.cs` to the `Infrastructure` folder and edit it as shown in listing 18.

Listing 18: `UrlExtensions.cs`

```
using Microsoft.AspNetCore.Http;

namespace SportsStore.Infrastructure
{

    public static class UrlExtensions
    {

        public static string PathAndQuery(this HttpRequest request) =>
            request.QueryString.HasValue
                ? $"{request.Path}{request.QueryString}"
                : request.Path.ToString();
    }
}
```

23. Edit the `_ViewImports.cshtml` by adding a `using` directive as shown in listing 19.

Listing 19: Adding a `using` directive to `_ViewImports.cshtml`

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

24. Add the buttons by editing `ProductSummary.cshtml` as shown in listig 20.

Listing 20: Edits to `ProductSumary.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␣badge-pill␣badge-primary" style="float:right">
                <small>@Model.Price.ToString("c")</small>
            </span>
        </h4>
    </div>
    <form id="@Model.ProductID" asp-action="AddToCart"
          asp-controller="Cart" method="post">
        <input type="hidden" asp-for="ProductID" />
        <input type="hidden" name="returnUrl"
               value="@ViewContext.HttpContext.Request.PathAndQuery()" />
```

```
        <span class="card-text p-1">
            @Model.Description
            <button type="submit"
                    class="btn btn-success btn-sm pull-right" style="float:right">
                Add To Cart
            </button>
        </span>
    </form>
</div>
```

25. In the `Startup.cs` file, make the following changes in the `ConfigureServices()` method, shown in listing 21.

Listing 21: Additios to `ConfigureServices()`

```
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddDbContext<ApplicationDbContext>(options =>
                options.UseSqlServer(
                    Configuration["Data:SportStoreProducts:ConnectionString"]));
            services.AddTransient<IProductRepository, EFProductRepository>();
            services.AddMvc();
            services.AddMemoryCache();
            services.AddSession();
        }
```

26. In the `Startup.cs` file, make the following changes in the `Configure()` method, shown in listing 22.

Listing 22: Addition to `Configure()`

```
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            app.UseDeveloperExceptionPage();
            app.UseStatusCodePages();
            app.UseStaticFiles();
            app.UseSession();
// remainder of method omitted
        }
```

27. Implement the `CartController.cs` in the `Controllers` folder by adding a new class file as shown in listing 23.

Listing 23: Implementing the `CartController.cs`

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

namespace SportsStore.Controllers
{
    public class CartController : Controller
    {
        private IProductRepository repository;
        public CartController(IProductRepository repo)
        {
```

```
            repository = repo;
        }
        public RedirectToActionResult AddToCart(int productId, string returnUrl)
        {
            Product product = repository.Products
                .FirstOrDefault(p => p.ProductID == productId);

            if (product != null)
            {
                Cart cart = GetCart();
                cart.AddItem(product, 1);
                SaveCart(cart);
            }
            return RedirectToAction("Index", new { returnUrl });
        }
        public RedirectToActionResult RemoveFromCart(int productId,
                string returnUrl)
        {
            Product product = repository.Products
                .FirstOrDefault(p => p.ProductID == productId);

            if (product != null)
            {
                Cart cart = GetCart();
                cart.RemoveLine(product);
                SaveCart(cart);
            }
            return RedirectToAction("Index", new { returnUrl });
        }
        private Cart GetCart()
        {
            Cart cart = HttpContext.Session.GetJson<Cart>("Cart") ?? new Cart();
            return cart;
        }
        private void SaveCart(Cart cart)
        {
            HttpContext.Session.SetJson("Cart", cart);
        }
    }
}
```

28. Create a new class file named `SessionExtensions.cs` in the `Infrastructure` folder and edit it as shown in listing 24.

Listing 24: `SessionExtensions.cs`

```
using Microsoft.AspNetCore.Http;
using Newtonsoft.Json;

namespace SportsStore.Infrastructure
{

    public static class SessionExtensions
    {

        public static void SetJson(this ISession session, string key, object value)
        {
```

```
            session.SetString(key, JsonConvert.SerializeObject(value));
        }

        public static T GetJson<T>(this ISession session, string key)
        {
            var sessionData = session.GetString(key);
            return sessionData == null
                ? default(T) : JsonConvert.DeserializeObject<T>(sessionData);
        }
    }
}
```

29. In the `Models/ViewModels` folder, add a new class file named `CartIndexViewModel.cs` and edit it as shown in listing 25.

Listing 25: `CartIndexViewModel.cs`

```
using SportsStore.Models;

namespace SportsStore.Models.ViewModels
{

    public class CartIndexViewModel
    {
        public Cart Cart { get; set; }
        public string ReturnUrl { get; set; }
    }
}
```

30. In the tyCartController.cs file, add the `using SportsStore.Models.ViewModels` directive and an `Index()` method as shown in listing 26.

Listing 26: Additions to `CartController.cs`

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

namespace SportsStore.Controllers
{

    public class CartController : Controller
    {
        private IProductRepository repository;

        public CartController(IProductRepository repo)
        {
            repository = repo;
        }

        public ViewResult Index(string returnUrl)
        {
            return View(new CartIndexViewModel
```

```
            {
                Cart = GetCart(),
                ReturnUrl = returnUrl
            });
        }

    //remainder of file omitted
}
```

31. Create a new folder, `Views/Cart`, and add a Razor View file named `Index.cshtml`, as shown in listing 27.

Listing 27: `Views/Cart/Index.cshtml`

```
@model CartIndexViewModel

<h2>Your cart</h2>
<table class="table table-bordered table-striped">
    <thead>
        <tr>
            <th>Quantity</th>
            <th>Item</th>
            <th class="text-right">Price</th>
            <th class="text-right">Subtotal</th>
        </tr>
    </thead>
    <tbody>
        @foreach (var line in Model.Cart.Lines) {
            <tr>
                <td class="text-center">@line.Quantity</td>
                <td class="text-left">@line.Product.Name</td>
                <td class="text-right">@line.Product.Price.ToString("c")</td>
                <td class="text-right">
                    @((line.Quantity * line.Product.Price).ToString("c"))
                </td>
            </tr>
        }
    </tbody>
    <tfoot>
        <tr>
            <td colspan="3" class="text-right">Total:</td>
            <td class="text-right">
                @Model.Cart.ComputeTotalValue().ToString("c")
            </td>
        </tr>
    </tfoot>
</table>

<div class="text-center">
    <a class="btn btn-primary" href="@Model.ReturnUrl">Continue shopping</a>
</div>
```

32. Start Without Debugging and try out your Online Store.