Build an ASP.NET Core Service, and App with Core 2.2 Two-Day Hand-On Lab

Lab 12

This lab is the second in a series that creates the ASP.NET Core web application. This lab walks you through finishing the Controllers (complete with routing). Prior to starting this lab, you must have completed Lab 11.

NOTE: All of the views will be created in the next lab.

Part 1: Finish the Controllers

Step 1: Update the CartController

1) Open the `CartController.cs` file in the `SpyStore.Hol.Mvc` project. Update the using statements to match the following:

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using AutoMapper;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using SpyStore.Hol.Models.Entities;
using SpyStore.Hol.Models.ViewModels;
using SpyStore.Hol.Mvc.Controllers.Base;
using SpyStore.Hol.Mvc.Models.ViewModels;
using SpyStore.Hol.Mvc.Support;
```

2) Add the route attribute for Controller/Action:

```
[Route("[controller]/[action]")]
public class CartController : BaseController
{
  //Omitted for brevity
}
```

3) Update the constructor to create an `AutoMapper` configuration and add a class level variable to hold the configuration.

   a) Create a mapping from `AddToCartViewModel` to a `ShoppingCartRecord`. Set the target Id to 0 (zero) and the `Timestamp` value to null.

   b) Create a mapping from `CartRecordWithProductInfo` to `CartRecordViewModel`

   c) Create a mapping from `Product` to `AddToCartViewModel`. Map the `ProductDetails` fields to the equivalent fields on the target.

```
readonly MapperConfiguration _config = null;
public CartController(SpyStoreServiceWrapper serviceWrapper, IConfiguration configuration)
  : base(configuration)
  {
    _serviceWrapper = serviceWrapper;
    _config = new MapperConfiguration(cfg =>
    {
      cfg.CreateMap<AddToCartViewModel, ShoppingCartRecord>().AfterMap((s, t) =>
      {
        t.Id = 0;
        t.TimeStamp = null;
      });
      cfg.CreateMap<CartRecordWithProductInfo, CartRecordViewModel>();
      cfg.CreateMap<ProductViewModel, AddToCartViewModel>()
         .ForMember(x => x.Description, x => x.MapFrom(src => src.Details.Description))
         .ForMember(x => x.ModelName, x => x.MapFrom(src => src.Details.ModelName))
         .ForMember(x => x.ModelNumber, x => x.MapFrom(src => src.Details.ModelNumber))
         .ForMember(x => x.ProductImage, x => x.MapFrom(src => src.Details.ProductImage))
         .ForMember(x => x.ProductImageLarge,
                    x => x.MapFrom(src => src.Details.ProductImageLarge))
         .ForMember(x => x.ProductImageThumb,
                    x => x.MapFrom(src => src.Details.ProductImageThumb));
    });
}
```

4) Add the `Index` method, which gets the list of cart records for the current customer. The retrieved `Customer` and `ShoppingCartRecords` are converted into a `CartViewModel`. Return the default View with the `ViewModel`:

```
[HttpGet]
public async Task<IActionResult> Index()
{
  ViewBag.Title = "Cart";
  ViewBag.Header = "Cart";
  CartWithCustomerInfo cartWithCustomerInfo =
    await _serviceWrapper.GetCartAsync(ViewBag.CustomerId);
  var mapper = _config.CreateMapper();
  var viewModel = new CartViewModel
  {
    Customer = cartWithCustomerInfo.Customer,
    CartRecords = mapper.Map<IList<CartRecordViewModel>>(cartWithCustomerInfo.CartRecords)
  };
  return View(viewModel);
}
```

5) The `AddToCart` `HttpGet` method builds an `AddToCartViewModel` from the select Product and sets the Quantity to 1. Also add the `HttpGet` route attribute with the `{productid}` route parameter:

```
[HttpGet("{productId}")]
public async Task<IActionResult> AddToCart(int productId, bool cameFromProducts = false)
{
  ViewBag.CameFromProducts = cameFromProducts;
  ViewBag.Title = "Add to Cart";
  ViewBag.Header = "Add to Cart";
  ViewBag.ShowCategory = true;
  var prod = await _serviceWrapper.GetOneProductAsync(productId);
  if (prod == null) return NotFound();
  var mapper = _config.CreateMapper();
  AddToCartViewModel cartRecord = mapper.Map<AddToCartViewModel>(prod);
  cartRecord.Quantity = 1;
  return View(cartRecord);
}
```

6) The `HttpPost` version of the `AddToCart` method uses Model Binding to accept an `AddToCartViewModel`. The `ValidateAntiForgeryToken` is used in conjunction with the Form Tag Helper (covered later). If the Model Binding is successful, the customer, product, and quantity are passed into the SpyStoreServiceWrapper. If there is an error, the error is added to the ModelState and the AddToCart view is reloaded. If there isn't an error, the method redirects to the Index page.

```
[HttpPost("{productId}"), ValidateAntiForgeryToken]
public async Task<IActionResult> AddToCart(int productId, AddToCartViewModel item)
{
  if (!ModelState.IsValid) return View(item);
  try
  {
    await _serviceWrapper.AddToCartAsync(ViewBag.CustomerId, productId, item.Quantity);
  }
  catch (Exception)
  {
    ModelState.AddModelError(string.Empty, "There was an error adding the item to the cart.");
    return View(item);
  }
  return RedirectToAction(nameof(CartController.Index));
}
```

7) The `Update` `HttpPost` method is called with using JQuery Ajax from the view. The `CartRecordViewModel` is converted to a `ShoppingCartRecord` then sent to the service. If the record no longer exists (e.g. the quantity is set to zero), and `EmptyResult` is returned. Otherwise, the `PartialView` is returned to refresh the cart page.

```
[HttpPost("{id}"), ValidateAntiForgeryToken]
public async Task<IActionResult> Update(int id, CartRecordViewModel record)
{
  var cartRecord = new ShoppingCartRecord
  {
    Id = record.Id,
    Quantity = record.Quantity,
    ProductId = record.ProductId,
    TimeStamp = record.TimeStamp,
    CustomerId = record.CustomerId
  };
  var item = await _serviceWrapper.UpdateShoppingCartRecord(id, cartRecord);
  if (item == null)
  {
    return new EmptyResult();
  }
  var mapper = _config.CreateMapper();
  CartRecordViewModel vm = mapper.Map<CartRecordViewModel>(item);
  return PartialView(vm);
}
```

8) Add the Delete Action method:

```
[HttpPost("{id}"), ValidateAntiForgeryToken]
public async Task<IActionResult> Delete(int id, ShoppingCartRecord item)
{
  await _serviceWrapper.RemoveCartItemAsync(id, item);
  return RedirectToAction(nameof(Index));
}
```

9) Add the Buy method:

```
[HttpPost, ValidateAntiForgeryToken]
public async Task<IActionResult> Buy(Customer customer)
{
  var orderId = await _serviceWrapper.PurchaseAsync(ViewBag.CustomerId, customer);
  return RedirectToAction(
    nameof(OrdersController.Details),
    nameof(OrdersController).Replace("Controller", ""),
    new { orderId });
}
```

Step 2: Update the OrdersController

1) Open the `OrdersController.cs` file in the `SpyStore.Hol.Mvc` project. Update the using statements to the following:

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using SpyStore.Hol.Models.Entities;
using SpyStore.Hol.Models.ViewModels;
using SpyStore.Hol.Mvc.Controllers.Base;
using SpyStore.Hol.Mvc.Support;
```

2) Add the route attribute for Controller/Action:

```
[Route("[controller]/[action]")]
public class OrdersController : BaseController
{
  //Omitted for brevity
}
```

3) Add the `Index` action method:

```
[HttpGet]
public async Task<IActionResult> Index()
{
  ViewBag.Title = "Order History";
  ViewBag.Header = "Order History";
  IList<Order> orders = await _serviceWrapper.GetOrdersAsync(ViewBag.CustomerId);
  return View(orders);
}
```

4) Add the Details Action method:

```
[HttpGet("{orderId}")]
public async Task<IActionResult> Details(int orderId)
{
  ViewBag.Title = "Order Details";
  ViewBag.Header = "Order Details";
  OrderWithDetailsAndProductInfo orderDetails =
    await _serviceWrapper.GetOrderDetailsAsync(orderId);
  if (orderDetails == null) return NotFound();
  return View(orderDetails);
}
```

Step 3: Update the ProductsController

1) Open the `ProductController.cs` file in the `SpyStore.Hol.Mvc` project. Update the using statements to the following:

```
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using SpyStore.Hol.Models.Entities;
using SpyStore.Hol.Models.ViewModels;
using SpyStore.Hol.Mvc.Controllers.Base;
using SpyStore.Hol.Mvc.Support;
```

2) Add the route attribute for Controller/Action:

```
[Route("[controller]/[action]")]
public class OrdersController : BaseController
{
  //Omitted for brevity
}
```

3) Add the `Featured` action method. This returns the `ProductList.cshtml` view to show just the featured products:

```
[HttpGet]
public async Task<IActionResult> Featured()
{
  ViewBag.Title = "Featured Products";
  ViewBag.Header = "Featured Products";
  ViewBag.ShowCategory = true;
  ViewBag.Featured = true;
  return View("ProductList", await _serviceWrapper.GetFeaturedProductsAsync());
}
```

4) Add the `Index` action method. This redirects to the `Featured` action method:

```
[Route("/")]
[Route("Products")]
[Route("Products/Index")]
[HttpGet]
public ActionResult Index()
{
  return RedirectToAction(nameof(Featured));
}
```

5) Add the `Details` action method. The method redirects to the `AddToCart` action of the `CartController` with the `ProductID`, and `CameFromProducts` route parameters.

```
public ActionResult Details(int id)
{
  return RedirectToAction(nameof(CartController.AddToCart),
    nameof(CartController).Replace("Controller", ""),
    new { productId = id, cameFromProducts = true });
}
```

6) Add the `ProductList` action method which returns the `Products` for a specific `Category`:

```
[HttpGet]
public async Task<IActionResult> ProductList(int id)
{
  var cat = await _serviceWrapper.GetCategoryAsync(id);
  ViewBag.Title = cat?.CategoryName;
  ViewBag.Header = cat?.CategoryName;
  ViewBag.ShowCategory = false;
  ViewBag.Featured = false;
  return View(await _serviceWrapper.GetProductsForACategoryAsync(id));
}
```

7) Add the `Search` action method which returns the `Products` that match the search criteria:

```
[Route("[controller]/[action]")]
[HttpPost("{searchString}")]
public async Task<IActionResult> Search(string searchString)
{
  ViewBag.Title = "Search Results";
  ViewBag.Header = "Search Results";
  ViewBag.ShowCategory = true;
  ViewBag.Featured = false;
  return View("ProductList", await _serviceWrapper.SearchAsync(searchString));
}
```

Part 2: Update the App for the Products Controllers

Step 1: Change the Default Route in the Startup.cs Configure() method

1) Update the default route to the following:

```
routes.MapRoute(
  name: "default",
  template: "{controller=Products}/{action=Index}/{id?}");
```

Step 2: Update the Home Page Link in the Layout

1) Open the `_Layout.cshtml` file in Views\Shared, change the controller for the home link to `Products` from `Home` as follows (make this change twice):

```
<a asp-area="" asp-controller="Products" asp-action="Index" class="navbar-
brand">SpyStore_HOL.MVC</a>
<!-- omitted for brevity -->
<li><a asp-area="" asp-controller="Products" asp-action="Index">Home</a></li>
```

Summary

In this lab you finished the Controllers, added attribute routing, and added the ViewModels. It won’t properly run until after completing the next lab, which updates and/or adds the views.

Next steps

In the next part of this tutorial series, you will create the Views for the application.