Build an EF and ASP.NET Core 3.0 App HOL

Lab 7

This lab walks you through adding the View Models and finishing the Controllers (complete with routing). Prior to starting this lab, you must have completed Lab 6.

Part 1: Add the ASP.NET Core ViewModels

Step 1: Copy the ViewModels

NOTE: Copy the Assets\Lab7\ViewModels folder into the Models folder of the SpyStore.Hol.Mvc project.

Part 2: Finish the Controllers

Step 1: Update the HomeController

1) Add the route attribute for Controller/Action:
   Note: The reserved keywords are in square brackets "[]", any custom route variables are in braces "{}"

```
[Route("[controller]/[action]")]
```

2) Update the `Index` action method. This serves as the default action for the controller:

```
[Route("/Home")]
[Route("/Home/Index")]
[HttpGet]
public ActionResult Index()
{
  return View();
}
```

Step 2: Update the ProductsController

1) Add the route attribute for Controller/Action:

```
[Route("[controller]/[action]")]
```

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

```
[HttpGet]
public IActionResult Featured()
{
  ViewBag.Title = "Featured Products";
  ViewBag.Header = "Featured Products";
  ViewBag.ShowCategory = true;
  ViewBag.Featured = true;
  return View("ProductList", _productRepo.GetFeaturedWithCategoryName());
}
```

3) Add the `Index` action method. This redirects to the `Featured` action method and also serves as the entry point into the application and the default action for the controller:

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

4) 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 });
}
```

5) Add the `ProductList` action method which returns the `ProductList.cshtml` view with all of the `Products` for a specific `Category`. The action method gets the instance of the `ICategoryRepo` from the DI container by using the `[FromServices]` attribute:

```
[HttpGet]
public IActionResult ProductList([FromServices]ICategoryRepo categoryRepo, int id)
{
  var cat = categoryRepo.Find(id);
  ViewBag.Title = cat?.CategoryName;
  ViewBag.Header = cat?.CategoryName;
  ViewBag.ShowCategory = false;
  ViewBag.Featured = false;
  return View(_productRepo.GetProductsForCategory(id));
}
```

6) Add the `Search` action method which returns the `ProductList.cshtml` view with the `Products` that match the search criteria:

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

Step 3: Update the OrdersController

1) Add the route attribute for Controller/Action:

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

2) Add the `Index` action method:

```
[Route("/Orders")]
[Route("/Orders/Index")]
[HttpGet]
public IActionResult Index()
{
  ViewBag.Title = "Order History";
  ViewBag.Header = "Order History";
  _orderRepo.Context.CustomerId = ViewBag.CustomerId;
  IList<Order> orders = _orderRepo.GetOrderHistory().ToList();
  return View(orders);
}
```

3) Add the Details Action method:
   NOTE: The `{orderId}` parameter in the `HttpGet` attribute is appended to the controller route

```
[HttpGet("{orderId}")]
public IActionResult Details(int orderId)
{
  ViewBag.Title = "Order Details";
  ViewBag.Header = "Order Details";
  OrderWithDetailsAndProductInfo orderDetails = _orderRepo.GetOneWithDetails(orderId);
  if (orderDetails == null) return NotFound();
  return View(orderDetails);
}
```

Step 4: Update the CartController

1) Add the route attribute for Controller/Action:

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

2) 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(IShoppingCartRepo shoppingCartRepo)
{
  _shoppingCartRepo = shoppingCartRepo;
  _config = new MapperConfiguration(
    cfg =>
    {
      cfg.CreateMap<AddToCartViewModel, ShoppingCartRecord>()
         .AfterMap((s, t) => { t.Id = 0; t.TimeStamp = null; });
      cfg.CreateMap<CartRecordWithProductInfo, CartRecordViewModel>();
      cfg.CreateMap<Product, 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));
    });
}
```

3) Update the `Index` method. First, add the `[HttpGet]` route attribute. In the action method, create a `CartViewModel` from the `Customer` and `ShoppingCart` repos. Return the default View with the `ViewModel`:

```
[Route("/Cart")]
[Route("/Cart/Index")]
[HttpGet]
public IActionResult Index([FromServices] ICustomerRepo customerRepo)
{
  ViewBag.Title = "Cart";
  ViewBag.Header = "Cart";
  IEnumerable<CartRecordWithProductInfo> cartItems =
    _shoppingCartRepo.GetShoppingCartRecords(ViewBag.CustomerId);
  var customer = customerRepo.Find(ViewBag.CustomerId);
  var mapper = _config.CreateMapper();
  var viewModel = new CartViewModel
  {
    Customer = customer,
    CartRecords = mapper.Map<IList<CartRecordViewModel>>(cartItems)
  };
  return View(viewModel);
}
```

4) 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 IActionResult AddToCart([FromServices] IProductRepo productRepo,
  int productId, bool cameFromProducts = false)
{
  ViewBag.CameFromProducts = cameFromProducts;
  ViewBag.Title = "Add to Cart";
  ViewBag.Header = "Add to Cart";
  ViewBag.ShowCategory = true;
  var prod = productRepo.GetOneWithCategoryName(productId);
  if (prod == null) return NotFound();
  var mapper = _config.CreateMapper();
  AddToCartViewModel cartRecord = mapper.Map<AddToCartViewModel>(prod);
  cartRecord.Quantity = 1;
  return View(cartRecord);
}
```

5) 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 `AddToCartViewModel` is converted to a `ShoppingCartRecord` class, then saved to the database. If anything fails on the save, the exception is added to the `ModelState` and the user gets to try again. Add the following code:

```
[HttpPost("{productId}"), ValidateAntiForgeryToken]
public IActionResult AddToCart(int productId, AddToCartViewModel item)
{
  if (!ModelState.IsValid) return View(item);
  try
  {
    var mapper = _config.CreateMapper();
    var cartRecord = mapper.Map<ShoppingCartRecord>(item);
    cartRecord.DateCreated = DateTime.Now;
    _shoppingCartRepo.Context.CustomerId = ViewBag.CustomerId;
    _shoppingCartRepo.Add(cartRecord);
  }
  catch (Exception)
  {
    ModelState.AddModelError(string.Empty, "There was an error adding the item to the cart.");
    return View(item);
  }
  return RedirectToAction(nameof(CartController.Index));
}
```

6) The `Update` `HttpPost` method is called with Ajax from the view. The ShoppingCartRecord is pulled from the database, and if concurrency checks pass, the quantity and time created are updated, and the record is saved to the database. If there are any issues, the errors are added to the ModelState, and the record is mapped to a `CartRecordViewModel`, then returned to the user.

```
[HttpPost("{id}"), ValidateAntiForgeryToken]
public IActionResult Update(CartRecordViewModel record)
{
  _shoppingCartRepo.Context.CustomerId = ViewBag.CustomerId;
  ShoppingCartRecord dbItem = _shoppingCartRepo.Find(record.Id);
  var mapper = _config.CreateMapper();
  if (record.TimeStamp != null && dbItem.TimeStamp.SequenceEqual(record.TimeStamp))
  {
    dbItem.Quantity = record.Quantity;
    dbItem.DateCreated = DateTime.Now;
    try
    {
      _shoppingCartRepo.Update(dbItem);
      CartRecordWithProductInfo updatedItem = _shoppingCartRepo.GetShoppingCartRecord(dbItem.Id);
      if (updatedItem == null)
      {
        return new EmptyResult();
      }
      CartRecordViewModel newItem = mapper.Map<CartRecordViewModel>(updatedItem);
      return PartialView(newItem);
    }
    catch (Exception)
    {
      ModelState.AddModelError(string.Empty, "An error occurred updating the cart.  Please reload
the page and try again.");
      var updatedItem = _shoppingCartRepo.GetShoppingCartRecord(dbItem.ProductId);
      CartRecordViewModel newItem = mapper.Map<CartRecordViewModel>(updatedItem);
      return PartialView(newItem);
    }
  }
  ModelState.AddModelError("", "Another user has updated the record.");
  CartRecordViewModel item = _shoppingCartRepo.GetShoppingCartRecord(dbItem.ProductId);
  var vm = mapper.Map<CartRecordViewModel>(dbItem);
  return PartialView(vm);
}
```

7) Add the Delete Action method:

```
[HttpPost("{id}"), ValidateAntiForgeryToken]
public IActionResult Delete(int customerId, int id, ShoppingCartRecord item)
{
  _shoppingCartRepo.Context.CustomerId = ViewBag.CustomerId;
  _shoppingCartRepo.Delete(item);
  return RedirectToAction(nameof(Index), new { customerId });
}
```

8) Add the Buy method:

```
[HttpPost, ValidateAntiForgeryToken]
public async Task<IActionResult> Buy()
{
  int orderId = _shoppingCartRepo.Purchase(ViewBag.CustomerId);
  return RedirectToAction(
    nameof(OrdersController.Details),
    nameof(OrdersController).Replace("Controller", ""),
    new { orderId });
}
```

Part 3: Remove the RouteTable

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

1) Change the call to app.UseEndpoints to not create a route table entry:

```
app.UseEndpoints(endpoints =>
{
  endpoints.MapControllers();
});
```

Summary

In this lab you finished the Controllers, added attribute routing, and added the ViewModels. It won't properly run until creating the next lab, which adds the views.

Next steps

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