Build an EF and ASP.NET Core 2.2 App HOL

Lab 8

This lab walks you through managing client-side libraries and creating the Views for the application. Prior to starting this lab, you must have completed Lab 7.

Part 1: Add the Images and CSS

Step 1: Add the images

1) Delete the images from the `wwwroot\Images` folder.
2) Add the images from the `2.2\Code\Completed\Lab8\Assets\Images` folder.

Step 2: Update the CSS for the site

1) Delete site.min.css and site.css from the `wwwroot\css` folder
2) Add the site.css file from the `2.2\Code\Completed\Lab8\Assets\css` folder.

Part 2: Manage Client-Side Libraries

Visual Studio:

Library Manager is installed with Visual Studio 2017 15.8 and later. Confirm the installation by opening Tools -> Extensions and Updates and searching for “Microsoft Library Manager”. If it’s not in the list of installed tools, search for it online in the Extensions and Updates dialog.

Visual Studio Code:

1) Install the Library Manager CLI Tooling:

```
Dotnet tool install -g Microsoft.Web.LibraryManager.CLI
```

Step 1: Delete the lib directory from the default template

1) Delete the `wwwroot\lib` folder. It will be replaced with files using library manager.

Step 2: Add the libman.json file

Visual Studio

1) Right click on the `SpyStore.Mvc` project and select Manage Client-Side Libraries. This adds the `libman.json` file to the root of the project. Right click on the `libman.json` file and select "Enable restore on build". This will prompt for you to allow another Nuget package (`Microsoft.Web.LibraryManager.Build`) to be restored into the project.

Visual Studio Code

1) Create a new libman.json file with the following command:

```
libman init
```

2) Add the library manager restore on build package:

```
dotnet add SpyStore.Hol.Mvc package Microsoft.Web.LibraryManager.Build
```

Step 3: Update the libman.json file

1) Update the `libman.json` file to the following JSON (this file can be copied from 2.2\Code\Completed\Lab8\Assets):

```
{
  //https://api.cdnjs.com/libraries/jquery?output=human
  //https://api.cdnjs.com/libraries?output=human
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "defaultDestination": "wwwroot/lib",
  "libraries": [
    {
      "library": "jquery@3.3.1",
      "destination": "wwwroot/lib/jquery",
      "files": ["jquery.js", "jquery.min.js"]
    },
    {
      "library": "jquery-validate@1.17.0",
      "destination": "wwwroot/lib/jquery-validation",
      "files": ["jquery.validate.js", "jquery.validate.min.js", "additional-methods.js",
"additional-methods.min.js"]
    },
    {
      "library": "jquery-validation-unobtrusive@3.2.10",
      "destination": "wwwroot/lib/jquery-validation-unobtrusive",
      "files": ["jquery.validate.unobtrusive.js", "jquery.validate.unobtrusive.min.js" ]
    },
    {
      "library": "font-awesome@4.7.0",
      "destination": "wwwroot/lib/fontawesome"
    },
```

```
    {
      "library": "twitter-bootstrap@4.1.3",
      "destination": "wwwroot/lib/bootstrap4",
      "files": [
        "css/bootstrap-grid.css",
        "css/bootstrap-grid.css.map",
        "css/bootstrap-grid.min.css",
        "css/bootstrap-grid.min.css.map",
        "css/bootstrap-reboot.css",
        "css/bootstrap-reboot.css.map",
        "css/bootstrap-reboot.min.css",
        "css/bootstrap-reboot.min.css.map",
        "css/bootstrap.css",
        "css/bootstrap.css.map",
        "css/bootstrap.min.css",
        "css/bootstrap.min.css.map",
        "js/bootstrap.bundle.js",
        "js/bootstrap.bundle.js.map",
        "js/bootstrap.bundle.min.js",
        "js/bootstrap.bundle.min.js.map",
        "js/bootstrap.js",
        "js/bootstrap.js.map",
        "js/bootstrap.min.js",
        "js/bootstrap.min.js.map"
      ]
    }
  ]
}
```

Step 4: Update the _Layout.cshml file

1) Update the stylesheet `environment` tags to the following:

```
<environment include="Development">
  <link rel="stylesheet" href="~/lib/bootstrap4/css/bootstrap.css"/>
  <link href="~/lib/fontawesome/css/font-awesome.css" rel="stylesheet"/>
</environment>
<environment exclude="Development">
  <link rel="stylesheet"
     href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"
     integrity="sha384-MCw98/SFnGE8fJT3GXwEOngsV7Zt27NXFoaoApmYm81iuXoPkFOJwJ8ERdknLPMO"
     crossorigin="anonymous"
     asp-fallback-href="~/lib/bootstrap4/css/bootstrap.min.css"
     asp-fallback-test-class="sr-only" asp-fallback-test-property="position"
       asp-fallback-test-value="absolute" />
  <link href="~/lib/fontawesome/css/font-awesome.min.css" rel="stylesheet" />
</environment>
```

2) Remove/Comment out the CookieConsentPartial.

```
@*<partial name="_CookieConsentPartial" />*@
```

3) Update the JavaScript `environment` tags to the following:

```
<environment include="Development">
  <script src="~/lib/jquery/jquery.js" asp-append-version="true"></script>
  <script src="~/lib/bootstrap4/js/bootstrap.js"></script>
</environment>
<environment exclude="Development">
  <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-3.3.1.min.js"
    asp-fallback-src="~/lib/jquery/jquery.min.js" asp-fallback-test="window.jQuery"
    crossorigin="anonymous"
    integrity="sha384-tsQFqpEReu7ZLhBV2VZlAu7zcOV+rXbYlF2cqB8txI/8aZajjp4Bqd+V6D5IgvKT">
  </script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/js/bootstrap.min.js"
    asp-fallback-src="~/lib/bootstrap4/js/bootstrap.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
    crossorigin="anonymous"
    integrity="sha384-ChfqqxuZUCnJSK3+MXmPNIyE6ZbWh2IMqE241rYiqJxyMiZ6OW/JmZQ5stwEULTy">
  </script>
</environment>
```

Step 5: Update the _ValidationScriptsPartial.cshml file

1) Update the entire partial to the following:

```
<environment include="Development">
  <script src="~/lib/jquery-validation/jquery.validate.js"></script>
  <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js"></script>
</environment>
<environment exclude="Development">
  <script src="https://ajax.aspnetcdn.com/ajax/jquery.validate/1.17.0/jquery.validate.min.js"
    asp-fallback-src="~/lib/jquery-validate/jquery.validate.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.validator" crossorigin="anonymous"
    integrity="sha384-rZfj/ogBloos6wzLGpPkkOr/gpkBNLZ6b6yLy4o+ok+t/SAKlL5mvXLr0OXNi1Hp">
  </script>
  <script src=
"https://ajax.aspnetcdn.com/ajax/jquery.validation.unobtrusive/3.2.9/jquery.validate.unobtrusive.m
in.js"
    asp-fallback-src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.validator &&
window.jQuery.validator.unobtrusive"
    crossorigin="anonymous"
    integrity="sha384-ifv0TYDWxBHzvAk2Z0n8R434FL1Rlv/Av18DXE43N/1rvHyOG4izKst0f2iSLdds">
  </script>
</environment>
```

Part 4: Create the Shared Views and Templates

Step 1: Update the _ViewImports.cshtml File

1) Open the `_ViewImports.cshtml` file in the Views folder. This file is loaded before any Views at or below the level of this file in the directory tree. This enables a central place to include all of the using statements for the Views. Update the using statements to match the following:

```
@using SpyStore.Hol.Mvc
@using SpyStore.Hol.Mvc.Models
@using SpyStore.Hol.Models.Entities
@using SpyStore.Hol.Models.ViewModels
@using System.Collections.Generic
@using Microsoft.AspNetCore.Mvc.Rendering
@using SpyStore.Hol.Mvc.Models.ViewModels
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

Step 2: Create the DateTime DisplayTemplate

1) Create a new folder named `DisplayTemplates` under the `Views\Shared` folder.
2) Add a Partial View named `DateTime.cshtml` in the new folder.

1) Clear out the existing code and replace it with the following:

```
@using System.Threading.Tasks
@model DateTime?
@if (Model == null)
{
    @:Unknown
}
else
{
  if (ViewData.ModelMetadata.IsNullableValueType)
  {
    @:@(Model.Value.ToString("d"))
  }
  else
  {
    @:@(((DateTime)Model).ToString("d"))
  }
}
```

Step 3: Create the AddToCartViewModel Editor Template

Views named for types will be rendered when an Html helper is used to edit or display a model (or model property) of that type.

1) Create a new folder named `EditorTemplates` under the `Views\Shared` folder.
2) Add a Partial View named `AddToCartViewModel.cshtml` in the new folder.

3) Clear out the existing code and replace it with the following:

```
@model AddToCartViewModel
@if (Model.Quantity == 0)
{
  Model.Quantity = 1;
}
<div class="card">
  <div class="card-body">
    <h1 class="d-block">@Html.DisplayFor(x => x.ModelName)</h1>
    <div class="col-sm-6 "><img src="@Url.Content($"~/images/{Model.ProductImageLarge}")" /></div>
    <h1 class="d-none">@Html.DisplayFor(x => x.ModelName)</h1>
    <div class="price-label">Price:</div><div>@Html.DisplayFor(x => x.CurrentPrice)</div>
    <div>Only @Html.DisplayFor(x => x.UnitsInStock) left.</div>
    <div></div>
    <div>@Html.DisplayFor(x => x.Description)</div>
    <ul>
      <li> <div>MODEL NUMBER:</div> @Html.DisplayFor(x => x.ModelNumber)</li>
      <li>
        <div>CATEGORY:</div>
        <a asp-controller="Products" asp-action="ProductList"
           asp-route-id="@Model.CategoryId">@Model.CategoryName</a>
      </li>
    </ul>
    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="TimeStamp" />
    <input type="hidden" asp-for="CategoryId" />
    <input type="hidden" asp-for="CategoryName" />
    <input type="hidden" asp-for="CustomerId" value="@ViewBag.CustomerId" />
    <input type="hidden" asp-for="ProductId" />
    <input type="hidden" asp-for="LineItemTotal" />
    <input type="hidden" asp-for="Description" />
    <input type="hidden" asp-for="ModelNumber" />
    <input type="hidden" asp-for="ModelName" />
    <input type="hidden" asp-for="ProductImage" />
    <input type="hidden" asp-for="ProductImageLarge" />
    <input type="hidden" asp-for="ProductImageThumb" />
    <input type="hidden" asp-for="UnitsInStock" />
    <input type="hidden" asp-for="CurrentPrice" />
    <div class="row">
      <label>QUANTITY:</label>
      <input asp-for="Quantity" class="form-control" />
      <input type="submit" value="Add to Cart" class="btn btn-primary" />
    </div>
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <span asp-validation-for="Quantity" class="text-danger"></span>
  </div>
</div>
```

Step 4: Create the AddToCart View

This view doubles as the Product Details view

1) Add a View named AddToCart.cshtml view Shared folder
2) Update the code to the following:

```
@model AddToCartViewModel
@{
    ViewData["Title"] = @ViewBag.Title;
}
<h3>@ViewBag.Header</h3>
<form method="post"
      asp-controller="Cart"
      asp-action="AddToCart"
      asp-route-customerId="@ViewBag.CustomerId"
      asp-route-productId="@Model.Id">
    @Html.EditorForModel()
</form>
@{
    if (ViewBag.CameFromProducts != null && ViewBag.CameFromProducts)
    {
        <div>
            <a href="#" onclick="window.history.go(-1); return false;">Back to List</a>
        </div>
    }
}
@section Scripts {
    @{
        await Html.RenderPartialAsync("_ValidationScriptsPartial");
    }
}
```

Part 5: Create the LoginView Partial

Step 1: Create the View

1) Add a new View named `LoginView` to the `Views\Shared` Folder
2) Update the markup to the following:

```
<ul class="nav navbar-nav">
  <li><a asp-controller="Cart" asp-action="Index" title="Shopping Cart" class="nav-link">
    Cart <span class="fa fa-shopping-cart"></span></a></li>
  <li><a asp-controller="Orders" asp-action="Index" title="Order History" class="nav-link">
    Orders <span class="fa fa-tag"></span></a></li>
  <li><a href="#" class="dropdown-toggle nav-link" data-toggle="dropdown">SEARCH
        <span class="fa fa-search"></span></a>
      <div class="dropdown-menu dropdown-menu-right bg-primary">
        <form asp-controller="Products" asp-action="Search"
            class="form-inline justify-content-end" role="search">
          <div class="input-group md-4">
            <label class="sr-only" for="searchString">Search</label>
            <input type="text" id="searchString" name="searchString"
                class="form-control" placeholder="SEARCH">
            <span class="input-group-append">
              <button class="btn btn-light" type="submit">
                 <span class="fa fa-search"></span>
              </button>
            </span>
          </div>
        </form>
      </div>
  </li>
</ul>
```

Step 2: Add the Login Partial to the Layout

1) Add the login partial view to the layout using the new Partial View Tag Helper:

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Products" asp-
action="Index">Home</a>
    </li>
    <li class="nav-item">
      <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-
action="Privacy">Privacy</a>
    </li>
  </ul>
  <partial name="LoginView"/>
</div>
```

Part 6: Create the Products Views and Templates

Step 1: Create the Product Display Template

1) Create a new folder named `Products` under the Views folder. Create a new folder named `DisplayTemplates` under the `Views\Products` folder.
2) Add a Partial View named `Product.cshtml` in the new folder. Clear out the existing code and replace it with the following:

```
@model Product
<div class="col-6 col-sm-4 col-md-3">
  <div class="product">
    <img src="@Url.Content($"~/images/{Model.Details.ProductImage}")"/>
    <div class="price">@Html.DisplayFor(x => x.CurrentPrice)</div>
    <div class="title-container">
      <h5>
        <a asp-controller="Products" asp-action="Details" asp-route-id="@Model.Id">
          @Html.DisplayFor(x => x.Details.ModelName)</a></h5>
      </div>
    <div><span class="model-number">Model Number:</span>
          @Html.DisplayFor(x => x.Details.ModelNumber)</div>
    @if (ViewBag.ShowCategory)
    {
      <a asp-controller="Products" asp-action="ProductList"
         asp-route-id="@Model.CategoryId" class="category">@Model.CategoryName</a><br/>
    }
      <a asp-controller="Cart" asp-action="AddToCart" asp-route-productId="@Model.Id"
         asp-route-cameFromProducts="true" class="btn btn-primary btn-cart">
         Add To Cart <span class="fa fa-shopping-cart"></span></a>
  </div>
</div>
```

Step 2: Create the ProductList View

1) Add a View named `ProductList.cshtml` in the `Views\Products` folder. Clear out the existing code and replace it with the following:

```
@model IList<Product>
@{ ViewData["Title"] = ViewBag.Title; }
<div class="jumbotron">
  @if (ViewBag.Featured != true)
  {
    <a asp-controller="Products" asp-action="Featured" class="btn btn-info btn-lg">View Featured
Products &raquo;</a>
  }
</div>
<h3>@ViewBag.Header</h3>
<div class="row">
  @for (int x = 0; x < Model.Count; x++)
  {
    var item = Model[x];
    @Html.DisplayFor(model => item)
  }
</div>
```

Part 7: Create the Orders Views

Step 1: Create the Index View

1) Create a new folder named `Orders` under the `Views` folder, add a View named `Index.cshtml` in the new folder. clear out the existing code and replace it with the following:

```
@model IList<Order>
<div class="page container">
  <div class="card-body">
    <h3>@ViewBag.Header</h3>
    @for (int x = 0; x < Model.Count; x++)
    {
      var item = Model[x];
      <div class="order-card-heading">
        <div class="row">
          <div class="col-sm-2">
            <label>Order Number</label>
            <a asp-action="Details" asp-route-orderId="@item.Id">
              @Html.DisplayFor(model => item.Id)
            </a>
          </div>
          <div class="col-sm-2">
            <label asp-for="@item.OrderDate"></label><br />
            @Html.DisplayFor(model => item.OrderDate)
          </div>
          <div class="col-sm-2">
            <label asp-for="@item.ShipDate"></label><br />
            @Html.DisplayFor(model => item.ShipDate)
          </div>
          <div class="col-sm-2">
            <label asp-for="@item.OrderTotal"></label><br />
            @Html.DisplayFor(model => item.OrderTotal)
          </div>
          <div class="col-sm-2 order-actions">
            <a asp-action="Details" asp-route-orderId="@item.Id"
              class="btn btn-primary">Order Details</a>
          </div>
        </div>
      </div>
    }
  </div>
</div>
```

Step 2: Create the Details View

1) Add a View named `Details.cshtml` in the `Views\Orders` folder. Clear out the existing code and replace it with the following (this file is a long one – you might want to copy it from 2.2\Labs\Completed\Assets\Orders):

```
@model OrderWithDetailsAndProductInfo
@{
    ViewData["Title"] = "Details";
}
<div class="card-body">
  <h3>@ViewBag.Header</h3>
  <div class="row top-row">
    <div class="col-sm-6">
      <label asp-for="OrderDate"></label>
      <strong>@Html.DisplayFor(model => model.OrderDate)</strong>
    </div>
    <div class="col-sm-6">
      <label asp-for="ShipDate"></label>
      <strong>@Html.DisplayFor(model => model.ShipDate)</strong>
    </div>
  </div>
  <div class="row">
    <div class="col-sm-6">
      <label>Billing Address:</label>
      <address>
        <strong>John Doe</strong><br>
        123 State Street<br>
        Whatever, UT 55555<br>
        <abbr title="Phone">P:</abbr> (123) 456-7890
      </address>
    </div>
    <div class="col-sm-6">
      <label>Shipping Address:</label>
      <address>
        <strong>John Doe</strong><br>
          123 State Street<br>
          Whatever, UT 55555<br>
          <abbr title="Phone">P:</abbr> (123) 456-7890
          <abbr title="Email">E:</abbr>John.Doe@domain.com
        </address>
    </div>
  </div>
  <div class="table-responsive">
    <table class="table table-bordered product-table">
      <thead>
        <tr>
          <th style="width: 70%;">Product</th>
          <th class="text-right">Price</th>
          <th class="text-right">Quantity</th>
          <th class="text-right">Total</th>
        </tr>
      </thead>
```

```
      <tbody>
      @for (int x = 0; x < Model.OrderDetails.Count; x++)
      {
        var item = Model.OrderDetails[x];
        <tr>
          <td>
            <div class="product-cell-detail">
              <img src="@Url.Content($"~/images/{item.ProductImageThumb}")" class="pull-left" />
              <a asp-controller="Products" asp-action="Details"
                  asp-route-id="@item.ProductId" class="h5">
                @Html.DisplayFor(model => item.ModelName)
              </a>
              <div class="small text-muted">@Html.DisplayFor(model => item.Description)</div>
            </div>
          </td>
          <td class="text-right">@Html.DisplayFor(model => item.UnitCost)</td>
          <td class="text-right">@Html.DisplayFor(model => item.Quantity)</td>
          <td class="text-right">@Html.DisplayFor(model => item.LineItemTotal)</td>
        </tr>
      }
    </tbody>
    <tfoot>
      <tr>
        <th> </th>
        <th> </th>
        <th> </th>
        <th class="text-right">@Html.DisplayFor(model => model.OrderTotal)</th>
      </tr>
    </tfoot>
  </table>
  </div>
  <div class="pull-right">
    <a asp-action="Index" class="btn btn-primary">Back to Order History</a>
  </div>
</div>
```

Part 8: Create the Cart Views and Templates

Step 1: Create the CartRecordViewModel EditorTemplate

1) Create a new folder named `Cart` under the `Views` folder. Create a new folder named `EditorTemplates` under the `Views\Cart` folder.
2) Add a Partial View named `CartRecordViewModel.cshtml` in the new folder, clear out the existing code and replace it with the following:

```
@model CartRecordViewModel
@{
    var formName = "updateCartForm" + Model.Id;
}
<form asp-controller="Cart" asp-action="Update" asp-route-id="@Model.Id" id="@formName"
method="post">
    <div asp-validation-summary="ModelOnly" class="text-danger"></div>
    <span asp-validation-for="Quantity" class="text-danger"></span>
    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="TimeStamp" />
    <input type="hidden" asp-for="CustomerId" />
    <input type="hidden" asp-for="UnitsInStock" />
    <input type="hidden" asp-for="ProductId" />
    <input type="hidden" asp-for="LineItemTotal" />
    <input asp-for="Quantity" class="cart-quantity text-right" />
    <button class="btn btn-link btn-sm" onClick="updateCart($('#@formName')[0],
'@Url.Action("Update", "Cart", new {id = @Model.Id})', @Model.Id);return false;">Update</button>
</form>
<form asp-controller="Cart" asp-action="Delete" asp-route-id="@Model.Id" id="deleteCartForm"
method="post">
    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="TimeStamp" />
    <button class="btn btn-link btn-sm">Remove</button>
</form>
```

Step 2: Create the Index View

1) Add a View named Index.cshtml in the Views\Cart folder.
2) Clear out the existing code and replace it with the following:

```
@model CartViewModel
@{
  ViewData["Title"] = "Index";
  var cartTotal = 0M;
}
<h3>@ViewBag.Header</h3>
<div>
  <div class="table-responsive">
    <table class="table table-bordered product-table">
      <thead>
        <tr>
          <th style="width: 70%;">Product</th>
          <th class="text-right">Price</th>
          <th class="text-right">Quantity</th>
          <th class="text-right">Available</th>
          <th class="text-right">Total</th>
        </tr>
      </thead>
      @for (var x = 0; x < Model.CartRecords.Count; x++)
      {
        var item = Model.CartRecords[x];
        cartTotal += item.LineItemTotal;
        <partial name="Update" model="item" />
      }
      <tfoot>
        <tr>
          <th></th>
          <th> </th>
          <th> </th>
          <th> </th>
          <th><span id="CartTotal">@Html.FormatValue(cartTotal, "{0:C2}")</span></th>
        </tr>
      </tfoot>
    </table>
    <form asp-controller="Cart" asp-action="Buy">
      <div class="pull-right"><button class="btn btn-primary">Checkout</button></div>
    </form>
  </div>
</div>
```

3) Add the following JavaScript for the ajax call

```
@section Scripts
{
  <partial Name="_ValidationScriptsPartial"/>
  <script language="javascript" type="text/javascript">
    function updateCart(form, url, id) {
      "option strict";
      var quantity = form.elements["Quantity"].value;
      var timeStamp = form.elements["TimeStamp"].value;
      var token = $('input[name="__RequestVerificationToken"]', form).val();
      var myData = { TimeStamp: timeStamp, Quantity: quantity };
      var dataWithAntiforgeryToken = $.extend(myData, { '__RequestVerificationToken': token });
      $.ajax({
        url: url,
        type: "POST",
        data: dataWithAntiforgeryToken,
        success: function(data) {
          $("#row_" + id).replaceWith(data);
        },
        error: function() {
          alert('An error occurred: Please reload the page and try again.');
        }
      })
      .done(updateCartPrice);
    }
    function getSum(total, num) {
      "use strict";
      return total + Math.round(num.innerText * 100) / 100;
    }
    function updateCartPrice() {
      "use strict";
      var list = $('span[id^="rawTotal"]');
      var total = $.makeArray(list).reduce(getSum, 0);
      $('#CartTotal')[0].innerText =
        '$' + parseFloat(total).toFixed(2).toString().replace(/(\d)(?=(\d\d\d)+(?!\d))/g, "$1,");
    }
    $(function() {
      updateCartPrice();
    });
  </script>
}
```

Step 3: Create the Update View

1) Add a View named `Update.cshtml` in the Views\Cart folder.
2) Clear out the existing code and replace it with the following:

```
@model CartRecordViewModel
<tr id="row_@Model.Id">
  <td>
    <div class="product-cell-detail">
      <img src="@Url.Content($"~/images/{Model.ProductImageThumb}")" class="pull-left" />
      <a class="h5" asp-controller="Products" asp-action="Details"
        asp-route-id="@Model.ProductId">@Html.DisplayFor(model => model.ModelName)</a>
      <div class="small text-muted">@Html.DisplayFor(model => model.CategoryName)</div>
      <div class="small text-muted d-none d-sm-block">
        @Html.DisplayFor(model => model.Description)
      </div>
    </div>
  </td>
  <td class="text-right"> @Html.DisplayFor(model => model.CurrentPrice) </td>
  <td class="text-right cart-quantity-row"> @Html.EditorForModel() </td>
  <td class="text-right"> @Html.DisplayFor(model => model.UnitsInStock) </td>
  <td class="text-right">
    <span id="rawTotal_@Model.ProductId" class="d-none">@Model.LineItemTotal</span>
    <span id="total_@Model.ProductId">@Html.DisplayFor(model => model.LineItemTotal)</span>
  </td>
</tr>
```

Part 9: Delete the Home Controller Index Method and View

1) Delete the `Index` action method in the `HomeController`.
2) Delete the `Index` view from the `Views\Home` folder.

Part 10: Bundle and Minify the JavaScript

Step 1: Update the _ViewImports.cshtml file

1) Before the @addTagHelper for Microsoft.AspNetCore.Mvc.TagHelpers, add the following line:

```
@addTagHelper *, WebOptimizer.Core
```

Step 2: Update the Startup.cs file

1) In the `Configure` method, add `app.UseWebOptimizer` *before* the `UseStaticFiles` call.

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
  //omitted for brevity

  app.UseWebOptimizer();
  app.UseStaticFiles();
  //omitted for brevity
}
```

Step 3: Minimize the JavaScript and CSS files

To minimize specific files or to create bundles, add configuration options into the AddWebOptimizer() method.

1) In the `ConfigureServices` method, add `services.AddWebOptimizer`. If not configured any farther, this automatically minimizes all JS and CSS files.
2) Add `MinifyCssFiles` and `MinifyJsFiles` to minimize the files. Wrap the call in an if statement to disable bundling and minification in the Development and Local environments, but minify and bundle in any other environments.

```
if (_env.IsDevelopment() || _env.EnvironmentName == "Local")
{
  services.AddWebOptimizer(false,false);
}
else
{
  services.AddWebOptimizer(options =>
  {
    options.MinifyCssFiles(); //Minifies all CSS files
    //options.MinifyJsFiles(); //Minifies all JS files
    options.MinifyJsFiles("js/site.js");
    //options.AddJavaScriptBundle("js/validations/validationCode.js", "js/validations/**/*.js");
  });
}
```

3) The validation scripts are going to be built in Lab 11:

Summary

The lab updated the CSS for the site, managed client side libraries, and added the Views and Templates.

Next steps

In the next part of this tutorial series, you will create the menu using a View Components.