Build an EF and ASP.NET Core 3.0 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) Add the images from the `Lab8\Assets\Images` folder to the `wwwroot\Images` folder.

Step 2: Update the CSS for the site

1) Delete site.css from the `wwwroot\css` folder
2) Add the site.css file from the `Lab8\Assets\css` folder to the `wwwroot\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 as a global tool:

```
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.

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 2: Add the libman.json file

Visual Studio

1) Right click on the `SpyStore.Hol.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) Copy the `libman.json` file from `Lab8\`Assets to the root of the SpyStore.Mvc project (this will overwrite the file you just created).

Step 4: Update the _ValidationScriptsPartial.cshtml file

1) Copy the `_ValidationScriptsPartial.cshtml file` from Lab8\Assets\Views\Shared, replacing the existion file from the template.

Part 3: Create the Shared Views and Templates

Step 1: Copy and Update the _Layout.cshtml file

1) Copy the `_Layout.cshtml` file from `Lab8\Assets\Views\Shared` into the `Views\Shared` folder of the SpyStore.Hol.Mvc project, overwriting the existing one. Update the `<environment>` tag helpers for the "Local" environment, as shown here:
NOTE: There are four `environment` tag helpers total in the page, two `include` and two `exclude`

```
<environment include="Development, Local">
</environment>
<environment exclude="Development, Local">
</environment>
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 2: 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 3: Create the DateTime DisplayTemplate

1) Create a new folder named `DisplayTemplates` under the `Views\Shared` folder. Add a Partial View named `DateTime.cshtml` in the new folder.
2) 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 4: Copy 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.

3) Create a new folder named `EditorTemplates` under the `Views\Shared` folder. Copy the AddToCartViewModel.cshtml file from the `Lab8\Assets\Views\Shared\EditorTemplates` into the new folder.

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 5: Create the AddToCart View

This view doubles as the Product Details view

4) Add a View named AddToCart.cshtml to the Views\Shared folder and 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-antiforgery="true"
      id="myForm">
    @Html.EditorForModel()
    <div asp-validation-summary="All" class="text-danger"></div>
</form>
@{
    if (ViewBag.CameFromProducts != null && ViewBag.CameFromProducts)
    {
        <div>
            <a href="#" onclick="window.history.go(-1); return false;">Back to List</a>
        </div>
    }
}
@section Scripts {
    @{
        <partial name="_ValidationScriptsPartial" />
    }
}
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Part 4: Create the LoginView Partial

Step 1: Create the View

1) Add a new View named `LoginView` to the `Views\Shared` folder. This view also contains the Search form. 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:

```
<header class="navbar navbar-expand-sm navbar-light bg-primary navbar-static-top">
  <div class="container">
    <div class="navbar-header">
      <!—omitted for brevity -->
    </div>
    <nav class="collapse navbar-collapse header-collapse" id="navbarSupportedContent">
      <ul class="nav navbar-nav ml-auto">
        <li class="nav-item dropdown categories-dropdown d-block d-lg-none">
           <!—omitted for brevity -->
        </li>
        <partial name="LoginView"/>
      </ul>
    </nav>
  </div>
</header>
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Part 5: Copy the Products and Order Views and Templates

These views contain many examples of Tag Helpers in action. Spend some time exploring the views after copying them into the project. Copy the folders Views\Products and Views\Orders from Lab8\Assets into the Views folder of the SpyStore.Hol.Mvc project.

Part 6: 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. 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;
  var ts = JsonSerializer.Serialize(Model.TimeStamp).Replace("\"","");
}
<nocache>
  <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" id="TimeStamp" name="TimeStamp" value=@ts />
    <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" id="TimeStamp" name="TimeStamp" value=@ts />
    <button class="btn btn-link btn-sm">Remove</button>
  </form>
</nocache>
```

2) In order to correctly serialize and deserialize the TimeStamp property (which is a Byte[] in C#), we convert it into JSON and then use that value in the form. Ajax calls and serialization/deserialization of Byte[] is problematic, and needs some help.

Step 2: Copy the Index View

Copy the `Index.cshtml` file from Lab8\Assets\Views\Cart into the Views\Cart folder. This view contains the AJAX call to update the cart. Take some time to explore the file.

All files copyright Phil Japikse (http://www.skimedic.com/blog)

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: 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
}
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

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. At this point, you can run the app and explore some of the features.

Next steps

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

All files copyright Phil Japikse (http://www.skimedic.com/blog)