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 Tag Helpers and Views for the application. Prior to starting this lab, you must have completed Lab 7.

Part 1: 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.

Step 2: Add the libman.json file

Visual Studio

1) Right click on the `AutoLot.Web` 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 AutoLot.Web package Microsoft.Web.LibraryManager.Build
```

Step 3: Update the libman.json file

1) Add the following to the `libman.json` file:

```
{
  "version": "1.0",
  "defaultProvider": "cdnjs",
  "defaultDestination": "wwwroot/lib",
  "libraries": [
    {
      "library": "jquery@3.5.1",
      "destination": "wwwroot/lib/jquery",
      "files": [ "jquery.js", "jquery.min.js" ]
    },
    {
      "library": "jquery-validate@1.19.1",
      "destination": "wwwroot/lib/jquery-validation",
      "files": [ "jquery.validate.js", "jquery.validate.min.js", "additional-methods.js",
"additional-methods.min.js" ]
    },
    {
      "library": "jquery-validation-unobtrusive@3.2.11",
      "destination": "wwwroot/lib/jquery-validation-unobtrusive",
      "files": [ "jquery.validate.unobtrusive.js", "jquery.validate.unobtrusive.min.js" ]
    },
    {
      "library": "font-awesome@5.13.0",
      "destination": "wwwroot/lib/fontawesome"
    },
    {
      "library": "twitter-bootstrap@4.5.0",
      "destination": "wwwroot/lib/bootstrap",
      "files": [
        "css/bootstrap.css",
        "js/bootstrap.bundle.js",
        "js/bootstrap.js",
      ]
    }
  ]
}
```

Step 4: Update the _ValidationScriptsPartial.cshtml file

1) Replace the content of the `_ValidationScriptsPartial.cshtml file` with the following:

```
<environment include="Development">
  <script src="~/lib/jquery-validation/jquery.validate.js" asp-append-version="true"></script>
  <script src="~/lib/jquery-validation-unobtrusive/jquery.validate.unobtrusive.js" asp-append-
version="true"></script>
</environment>
<environment exclude="Development">
  <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.19.1/jquery.validate.min.js"
    asp-fallback-src="~/lib/jquery-validation/jquery.validate.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.validator"
    crossorigin="anonymous"
   integrity="sha256-F6h55Qw6sweK+t7SiOJX+2bpSAa3b/fnlrVCJvmEj1A=">
  </script>
  <script
    src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-
unobtrusive/3.2.11/jquery.validate.unobtrusive.min.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="sha256-9GycpJnliUjJDVDqP0UEu/bsm9U+3dnQUH8+3W10vkY=">
  </script>
</environment>
```

Part 2: Bundle and Minify the JavaScript/CSS

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 provides a central place to include all of the using statements for the Views. Update the using statements to match the following:

```
@using AutoLot.Web
@using AutoLot.Web.Models
@using AutoLot.Models.Entities
@using AutoLot.Web.ConfigSettings
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

2) To use WebOptimizer, the following line must be added:

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

3) To use any custom Tag Helpers (developed later in this lab), the following line must be added:

```
@addTagHelper *, AutoLot.Web
```

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. There are also methods (`MinifyCssFiles` and `MinifyJsFiles) that will` minimize the CSS and JS files respectively.

1) Add to the following code. The if block disables bundling and minification if the app is in development.

```
if (_env.IsDevelopment())
{
  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");
  });
}
```

Part 3: Add the Razor Syntax View

1) Add a new view named RazorSyntaxView to the Views\Home folder. Update the markup to the following:

```
@model Car
@{
    ViewData["Title"] = "RazorSyntax";
}

<h1>Razor Syntax</h1>
@{
    //Code Block
    var foo = "Foo";
    var bar = "Bar";
    var htmlString = "<ul><li>one</li><li>two</li></ul>";
}
```

```
@foo<br />
@htmlString<br />
@foo.@bar<br />
@Html.Raw(htmlString)
<hr />
@for (int i = 0; i < 15; i++)
{
}
@{
    @:Straight Text
    <div>Value:@Model.Id</div>
    <text>
        Lines without HTML tag
    </text>
    <br />
}
<hr/>
@*
    Multiline Comments
    Hi.
*@
Email Address Handling:<br/>
foo@foo.com = foo@foo.com<br/>
test@foo = test@foo<br/>
test@(foo) = testFoo<br/>
<hr/>
@functions {
    public static IList<string> SortList(IList<string> strings)  {
        var list = from s in strings orderby s select s;
        return list.ToList();
    }
}
-------------------------------------------------------------
@{ var myList = new List<string>  {"C", "A", "Z", "F"};
    var sortedList = SortList(myList); //MyFunctions.SortList(myList)
}
@foreach (string s in sortedList)
{
    @s@:&nbsp;
}
<hr/>
@{
    Func<dynamic, object> b = @<strong>@item</strong>;
}
This will be bold: @b("Foo")

@Html.DisplayForModel()
<hr/>
@Html.EditorForModel()
<hr/>
<a asp-controller="Cars" asp-action="Details" asp-route-id="@Model.Id">@Model.PetName</a>
```

Part 4: Create the Layout Partial Views

Step 1: Update the _Layout.cshtml file

1) Update the _Layout.cshtml view (located in Views\Shared) to the following:
   Note: The app won't work until you finish the remaining partials in this section.

```
@using Microsoft.AspNetCore.Hosting
@using Microsoft.Extensions.Hosting
@inject IWebHostEnvironment Env
<!DOCTYPE html>
<html lang="en">
<head>
  <partial name="Partials/_Head"/>
</head>
<body>
  <header>
    @{
      if (Env.IsDevelopment() || Env.IsEnvironment("Local"))
      {
        <p class="lead">Development</p>
      }
    }
    <partial name="Partials/_Menu"/>
  </header>
  <div class="container">
    <main role="main" class="pb-3">
      @RenderBody()
    </main>
  </div>
  <footer class="border-top footer text-muted">
    <div class="container">&copy; 2020 - AutoLot.Web -
      <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
  </footer>
  <partial name="Partials/_JavaScriptFiles"/>
  @RenderSection("Scripts", required: false)
</body>
</html>
```

Step 2: Create the _Head partial view

2) Create a new folder named `Partials` under the `Views\Shared` folder. Add a new view named _Head.cshtml to the folder. Update the markup to the following:

```
<meta charset="utf-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<title>@ViewData["Title"] - AutoLot.Web</title>
<environment include="Development,Local">
    <link rel="stylesheet" href="~/lib/bootstrap/css/bootstrap.css" asp-append-version="true" />
</environment>
<environment exclude="Development,Local">
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
          asp-fallback-href="~/lib/bootstrap/css/bootstrap.css"
          asp-fallback-test-class="sr-only" asp-fallback-test-property="position" asp-fallback-
test-value="absolute"
          crossorigin="anonymous"
          integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" />
</environment>
<link rel="stylesheet" href="~/lib/fontawesome/css/all.css" asp-append-version="true" />
<link rel="stylesheet" href="~/css/site.css" asp-append-version="true" />
```

Step 3: Create the _Menu partial view

1) Add a new view named _Menu.cshtml to the Views\Shared\Partials folder. Update the markup to the following:

```
@using Microsoft.Extensions.Options
@inject IOptionsMonitor<DealerInfo> DealerInfoMonitor
@{
  var dealerInfo = DealerInfoMonitor.CurrentValue;
}
<nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-
shadow mb-3">
  <div class="container">
    <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">@dealerInfo.DealerName</a>
    <button class="navbar-toggler" type="button" data-toggle="collapse"
        data-target=".navbar-collapse" aria-controls="navbarSupportedContent"
        aria-expanded="false" aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>
    <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
      <ul class="navbar-nav flex-grow-1">
        <li class="nav-item dropdown">
          <a class="nav-link dropdown-toggle text-dark" data-toggle="dropdown">
            Inventory <i class="fa fa-car"></i></a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-dark" asp-area=""
            asp-controller="Home" asp-action="RazorSyntax">Razor Syntax</a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-dark" asp-area="" asp-controller="Home"
            asp-action="Privacy">Privacy</a>
        </li>
      </ul>
    </div>
  </div>
</nav>
```

Step 4: Create the _JavaScriptFiles partial view

1) Create a new view named _JaveScriptFiles.cshtml under the `Views\Shared\Partials` folder. Update the markup to the following:

```
<environment include="Development,Local">
  <script src="~/lib/jquery/jquery.js" asp-append-version="true"></script>
  <script src="~/lib/bootstrap/js/bootstrap.bundle.js" asp-append-version="true"></script>
</environment>
<environment exclude="Development,Local">
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"
    asp-fallback-src="~/lib/jquery/jquery.min.js" asp-fallback-test="window.jQuery"
    crossorigin="anonymous"
    integrity="sha256-FgpCb/KJQlLNfOu91ta32o/NMZxltwRo8QtmkMRdAu8=">
  </script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.bundle.min.js"
    asp-fallback-src="~/lib/bootstrap/js/bootstrap.bundle.min.js"
    asp-fallback-test="window.jQuery && window.jQuery.fn && window.jQuery.fn.modal"
    crossorigin="anonymous"
    integrity="sha384-xrRywqdh3PHs8keKZN+8zzc5TX0GRTLCcmivcbNJWm2rs5C8PRhcEn3czEjhAO9o">
  </script>
</environment>
<script src="~/js/site.js" asp-append-version="true"></script>
```

Part 5: Create the DateTime Display Template

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

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

Part 5: Add the Custom TagHelpers

This site uses three custom Tag Helpers and a base class for common logic shared between the three.

Step 1: Create the RemoveController string extension

1) Create a new directory named `Extensions` in the AutoLot.Web project. In this directory create a new class named StringExtensions.cs. Make the class static and public, and add a using for System:

```
using System;
public static class StringExtensions
{
}
```

2) Add a static method to remove the word "Controller" from the end of a string:

```
public static string RemoveController(this string original)
  => original.Replace("Controller", "", StringComparison.OrdinalIgnoreCase);
```

Step 2: Create the CarLinkTagHelperBase class

1) Create a new directory named `TagHelpers` in the AutoLot.Web project. In this directory create a new class named CarLinkTagHelperBase.cs. Add the following using statements to the class:

```
using Microsoft.AspNetCore.Mvc.Infrastructure;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Razor.TagHelpers;
```

2) Make the class public and abstract, inherit from TagHelper, and create a public property named CarId:

```
public abstract class CarLinkTagHelperBase : TagHelper
{
  public int CarId { get; set; }
}
```

3) Add a constructor that takes an IActionContextAccessor and an IUrlHelperFactory and assigns them to class level variables:

```
protected readonly IUrlHelper UrlHelper;
protected CarLinkTagHelperBase(
  IActionContextAccessor contextAccessor, IUrlHelperFactory urlHelperFactory)
{
  UrlHelper = urlHelperFactory.GetUrlHelper(contextAccessor.ActionContext);
}
```

4) Create a protected method to build out the content:

```
protected void BuildContent(TagHelperOutput output, string actionName, string className,
  string displayText, string fontAwesomeName)
{
  output.TagName = "a"; // Replaces <email> with <a> tag
  var target =
    UrlHelper.Action(actionName, nameof(CarsController).RemoveController(), new {id = CarId});
  output.Attributes.SetAttribute("href", target);
  output.Attributes.Add("class",className);
  output.Content.AppendHtml($@"{displayText} <i class=""fas fa-{fontAwesomeName}""></i>");
}
```

Step 3: Create the CarDetailsTagHelper class

1) Add a class named `CarDetailsTagHelper.cs` into the TagHelpers directory. Make the class public and inherit CarLinkTagHelperBase. Add the following using statements to the class:

```
using AutoLot.Web.Controllers;
using Microsoft.AspNetCore.Mvc.Infrastructure;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Razor.TagHelpers;
```

2) Create a public constructor that passes the IActionContextAccessor and IUrlHelperFactory into the base class:

```
public class CarDetailsTagHelper : CarLinkTagHelperBase
{
  public CarDetailsTagHelper(
    IActionContextAccessor contextAccessor, IUrlHelperFactory urlHelperFactory)
    : base(contextAccessor, urlHelperFactory) { }
}
```

3) Override the Process method, passing in a controller action name, CSS class name, display text, and Font Awesome icon:

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
  BuildContent(output,nameof(CarsController.Details),"text-info","Details","info-circle");
}
```

Step 4: Create the DeleteCarTagHelper class

1) Add a class named Delete`CarTagHelper.cs` into the TagHelpers directory. Make the class public and inherit CarLinkTagHelperBase. Add the following using statements to the class:

```
using AutoLot.Web.Controllers;
using Microsoft.AspNetCore.Mvc.Infrastructure;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Razor.TagHelpers;
```

2) Create a public constructor that passes the IActionContextAccessor and IUrlHelperFactory into the base class:

```
public class CarDetailsTagHelper : CarLinkTagHelperBase
{
  public CarDetailsTagHelper(
    IActionContextAccessor contextAccessor, IUrlHelperFactory urlHelperFactory)
    : base(contextAccessor, urlHelperFactory) { }
}
```

3) Override the Process method, passing in a controller action name, CSS class name, display text, and Font Awesome icon:

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
  BuildContent(output,nameof(CarsController.Delete),"text-danger","Delete","trash");
}
```

Step 5: Create the EditCarTagHelper class

4) Add a class named Edit`CarTagHelper.cs` into the TagHelpers directory. Make the class public and inherit CarLinkTagHelperBase. Add the following using statements to the class:

```
using AutoLot.Web.Controllers;
using Microsoft.AspNetCore.Mvc.Infrastructure;
using Microsoft.AspNetCore.Mvc.Routing;
using Microsoft.AspNetCore.Razor.TagHelpers;
```

5) Create a public constructor that passes the IActionContextAccessor and IUrlHelperFactory into the base class:

```
public class CarDetailsTagHelper : CarLinkTagHelperBase
{
  public CarDetailsTagHelper(
    IActionContextAccessor contextAccessor, IUrlHelperFactory urlHelperFactory)
    : base(contextAccessor, urlHelperFactory) { }
}
```

6) Override the Process method, passing in a controller action name, CSS class name, display text, and Font Awesome icon:

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
  BuildContent(output,nameof(CarsController.Edit),"text-warning","Edit","edit");
}
```

Part 6: Add the Car Views and Templates

Step 1: Add the Car EditorTemplate

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

```
@model AutoLot.Models.Entities.Car

<div class="form-group">
  <label asp-for="MakeId" class="control-label"></label>
  <select asp-for="MakeId" class="form-control" asp-items="ViewBag.MakeId"></select>
</div>
<div class="form-group">
  <label asp-for="Color" class="control-label"></label>
  <input asp-for="Color" class="form-control"/>
  <span asp-validation-for="Color" class="text-danger"></span>
</div>
<div class="form-group">
  <label asp-for="PetName" class="control-label"></label>
  <input asp-for="PetName" class="form-control"/>
  <span asp-validation-for="PetName" class="text-danger"></span>
</div>
```

Step 2: Add the Car DisplayTemplate

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

```
@model AutoLot.Models.Entities.Car
<dl class="row">
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.MakeId)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.MakeNavigation.Name)
  </dd>
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.Color)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.Color)
  </dd>
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.PetName)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.PetName)
  </dd>
  <dt class="col-sm-2">
    @Html.DisplayNameFor(model => model.TimeStamp)
  </dt>
  <dd class="col-sm-10">
    @Html.DisplayFor(model => model.TimeStamp)
  </dd>
</dl>
```

Step 3: Add the Index View and CarList Partial View

1) Add a new directory named Partials to the Views\Car folder. In this directory, create a partial view named _CarListPartial.cshtml.

2) Clear out the existing markup and replace it with this:

```
@model IEnumerable<AutoLot.Models.Entities.Car>
@{
  var showMake = true;
  if (bool.TryParse(ViewBag.ByMake?.ToString(), out bool byMake))
  {
    showMake = !byMake;
  }
}
<p>
  <a asp-action="Create">Create New</a>
</p>
<table class="table">
  <thead>
    <tr>
      @if (showMake)
      {
        <th>@Html.DisplayNameFor(model => model.MakeId) </th>
      }
      <th>@Html.DisplayNameFor(model => model.Color) </th>
      <th>@Html.DisplayNameFor(model => model.PetName) </th>
      <th>@Html.DisplayNameFor(model => model.TimeStamp) </th>
      <th></th>
    </tr>
  </thead>
  <tbody>
  @foreach (var item in Model)
  {
    <tr>
    @if (showMake)
    {
      <td>@Html.DisplayFor(modelItem => item.MakeNavigation.Name)</td>
    }
      <td>@Html.DisplayFor(modelItem => item.Color)</td>
      <td>@Html.DisplayFor(modelItem => item.PetName)</td>
      <td>@Html.DisplayFor(modelItem => item.TimeStamp)</td>
      <td>
        <edit-car car-id="@item.Id"></edit-car> |
        <car-details car-id="@item.Id"></car-details> |
        <delete-car car-id="@item.Id"></delete-car>
      </td>
    </tr>
  }
  </tbody>
</table>
```

3) Add a View named `Index.cshtml` in the Views\Car folder.

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

```
@model IEnumerable<AutoLot.Models.Entities.Car>
@{
    ViewData["Title"] = "Index";
}
<h1>Vehicle Inventory</h1>
<partial name="Partials/_CarListPartial" model="@Model"/>
```

Step 4: Add the Details View

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

```
@model AutoLot.Models.Entities.Car
@{
  ViewData["Title"] = "Details";
}
<h1>Details</h1>
<hr />
<div>
  @Html.DisplayForModel()
</div>
<div>
  <edit-car car-id="@Model.Id"></edit-car>
  <a asp-action="Index">Back to List</a>
</div>
```

Step 5: Add the Create View

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

```
@model AutoLot.Models.Entities.Car
@{
    ViewData["Title"] = "Create";
}
<h1>Create</h1>
<hr />
<div class="row">
  <div class="col-md-4">
    <form asp-action="Create">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      @Html.EditorForModel()
      <div class="form-group">
        <input type="submit" value="Create" class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
<div>
  <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Step 6: Add the Edit View

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

```
@model AutoLot.Models.Entities.Car
@{
    ViewData["Title"] = "Edit";
}
<h1>Edit</h1>
<hr />
<div class="row">
  <div class="col-md-4">
    <form asp-area="" asp-controller="Cars" asp-action="Edit2" asp-route-id="@Model.Id">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      @Html.EditorForModel()
      <input type="hidden" asp-for="Id" />
      <input type="hidden" asp-for="TimeStamp" />
      <div class="form-group">
        <input type="submit" value="Save" class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
<div>
  <a asp-action="Index">Back to List</a>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Step 7: Add the Delete View

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

```
@model AutoLot.Models.Entities.Car
@{
    ViewData["Title"] = "Delete";
}
<h1>Delete</h1>
<h3>Are you sure you want to delete this?</h3>
<div>
  <h4>Car</h4>
  <hr />
  @Html.DisplayForModel()
  <form asp-action="Delete">
    <input type="hidden" asp-for="Id" />
    <input type="hidden" asp-for="TimeStamp" />
    <input type="submit" value="Delete" class="btn btn-danger" /> |
    <a asp-action="Index">Back to List</a>
  </form>
</div>
```

Step 8: Add the ByMake View

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

```
@model IEnumerable<AutoLot.Models.Entities.Car>
@{
  ViewData["Title"] = "Index";
}
<h1>Vehicle Inventory for @ViewBag.MakeName</h1>
@{
  var mode = new ViewDataDictionary(ViewData) {{"ByMake", true}};
}
<partial name="Partials/_CarListPartial" model="Model" view-data="@mode"/>
```

Summary

The lab updated the CSS for the site, managed client-side libraries, and added the Views and Templates. You can explore the Car views by running the app and adding Cars/Index to the route.

Next steps

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