.NET 10 App Dev Hands-On Lab

Razor Pages Lab 6 – Razor Pages

This lab walks you through creating the `RazorSyntax` page, the `BasePageModel`, and the `Cars` pages. Prior to starting this lab, you must have completed Razor Pages Lab 5.

Part 1: Add the Razor Syntax Page

- Add the following to the `GlobalUsings.cs` file:

```
global using Microsoft.AspNetCore.Mvc.Rendering;
```

- Add a new page named `RazorSyntax` to the `Pages` folder. Update the code-behind file to the following:

```
namespace AutoLot.Web.Pages;

public class RazorSyntaxModel(ICarRepo repo, IMakeRepo makeRepo) : PageModel
{
  [ViewData]
  public SelectList LookupValues { get; set; } =
    new(makeRepo.GetAll(), nameof(Make.Id), nameof(Make.Name));
  [ViewData]
  public string Title => "Razor Syntax";
  [BindProperty]
  public Car Entity { get; set; }
  public IActionResult OnGet()
  {
    Entity = repo.Find(6);
    return Page();
  }
}
```

- Update the view markup to the following:

```
@page
@model AutoLot.Web.Pages.RazorSyntaxModel
@{
  //can be set here or in the code behind
  //ViewData["Title"] = "Razor Syntax";
}
<h1>Razor Syntax</h1>
@for (int i = 0; i < 15; i++) { /*do something here */ }
@{
  //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 />
@foo.ToUpper()<br/>
@Html.Raw(htmlString)<br/>
```

```
<hr />
@{
  @:Straight Text
  <div>Value:@Model.Entity.Id</div>
  <text>
    Lines without HTML tag
  </text>
  <br />
}
<hr/>
Email Address Handling:<br/>
foo@foo.com<br />
@@foo<br/>
test@foo<br/>
test@(foo)<br />
@*
  Multiline Comments
  Hi.
*@
@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);
}
@foreach (string s in sortedList)
{
  @s@: 
}
<hr/>
@{
  Func<dynamic, object> b = @<strong>@item</strong>;
}
This will be bold: @b("Foo")
<hr/>
The Car named @Model.Entity.PetName is a <span
style="color:@Model.Entity.Color">@Model.Entity.Color</span> @Model.Entity.MakeNavigation.Name
<hr/>
Display For examples
Make:
@Html.DisplayFor(x=>x.Entity.MakeNavigation)
Car:
<div class="container">
    @Html.DisplayFor(c=>c.Entity)
</div>
Car Editor:
@Html.EditorFor(c=>c.Entity)
```

- Update the `_Menu.cshtml` partial for the new page (place it after the closing `</li>` tag for the inventory menu drop-down):

```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-page="/RazorSyntax">
    Razor Syntax <i class="fa-solid fa-cut"></i>
  </a>
</li>
```

Part 2: Add the SimpleService Page

- Add a new page named `SimpleService` to the `Pages` folder. Update the code-behind file to the following:

```
namespace AutoLot.Web.Pages;

public class SimpleServiceModel : PageModel
{
  public string Message { get; set; }
  public void OnGetServiceOne([FromKeyedServices(nameof(SimpleServiceOne))]ISimpleService service)
  {
    Message = service.SayHello();
  }
  public void OnGetServiceTwo([FromKeyedServices(nameof(SimpleServiceTwo))]ISimpleService service)
  {
    Message = service.SayHello();
  }
}
```

- Update the view markup to the following:

```
@page
@model AutoLot.Web.Pages.SimpleServiceModel
<h1>@Model.Message</h1>
```

- Update the `_Menu.cshtml` partial for the new page, adding the new menu items after the Razor Syntax menu item:

```
<li class="nav-item dropdown">
  <a class="nav-link dropdown-toggle text-dark" data-bs-toggle="dropdown">
    DI <i class="fa fa-syringe"></i>
  </a>
  <div class="dropdown-menu">
    <a class="dropdown-item text-dark" asp-page="/SimpleService" asp-page-handler="ServiceOne">
      Service One <i class="fa-solid fa-1 fa-xs" style="color: #74C0FC;"></i>
    </a>
    <a class="dropdown-item text-dark" asp-page="/SimpleService" asp-page-handler="ServiceTwo">
      Service Two <i class="fa-solid fa-2 fa-xs" style="color: #74C0FC;"></i>
    </a>
  </div>
</li>
```

Part 3: Create the BasePageModel

Copilot Agent Mode

These prompts complete Part 3 of this lab.

```
Setup Prompt: Always use file scoped namespaces. Always combine attributes on a single line when
possible. The project does not use nullable reference types. There is a GlobalUsings.cs file that
includes common usings, don't include using statements in new files if they are already in the
globalusings.cs file. I prefer expression bodied members when possible. Single line if statements
should still use braces. Use ternary operators when appropriate.
Use internal over private. All classes and methods are public unless told otherwise.
Use primary constructors when possible and don't declare a class level variable if the parameter
from the constructor can be used. All work is to be done in the AutoLot.Web project unless
otherwise specified.

Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Dal.Repos.Base;
global using AutoLot.Dal.Repos.Interfaces.Base;
global using AutoLot.Models.Entities;
global using AutoLot.Models.Entities.Base;
global using Microsoft.AspNetCore.Mvc.Rendering;

Prompt: In the Pages folder, add a new folder named Base and add a new public abstract class named
BasePageModel that inherits from PageModel. Make it generic with TEntity, constrained to
BaseEntity, new(). Accept three parameters in the constructor, IAppLogging (appLoggingInstance)
and IBaseRepo<TEntity> (baseRepoInstance) and string pageTitle. Assign the first parameters to two
protected readonly properties named AppLoggingInstance and BaseRepoInstance, respectively.
Assign the pageTitle to a public property {get; init:} named Title with the ViewData attribute.
Add a public property named Entity of type TEntity with get and set accessors and the BindProperty
attribute.
Add the following public properties:
SelectList LookupValues {get; set;}
string Error {get; set;}
Add a protected virtual method named GetLookupValues (input:none. return type: void) that sets the
LookupValues to null.
Add the four Crud methods:
protected virtual void GetOne Entity (input:int? id. Logic: if  id is null, make Entity null and
Error = "Invalid Request". Then return.
Otherwise assign Entity to the result of calling Find on BaseRepoInstance with id. If returned
value is null, set Error = "Not found", else Error = string.Empty)
protected virtual IActionResult SaveOne (intput: Func<TEntity,bool,int> saveFunction. Logic: if
ModelState is not valid, return Page().
If it is valid, call saveFunction passing in true for persist. After saving the record, redirect
to Details page)
protected virtual IActionResult SaveOneWithLookup (intput: Func<TEntity,bool,int> saveFunction.
Logic: if ModelState is not valid, call GetLookupValues() and return Page().
If it is valid, call saveFunction passing in true for persist. After saving the record, redirect
to Details page)
protected virtual IActionResult DeleteOne (input: int id. Logic: Call Delete on BaseRepoInstance
with Entity then redirect to Index)
```

```
Prompt: Add the following global usings to the GlobalUsings.cs file if they do not already exist
(sorted alphabetically. Don't remove any existing global using statements).
global using AutoLot.Web.Pages.Base;
```

Manual

Step 1: Create the BasePageModel class, constructor, and helper methods

- Add the following to the `GlobalUsings.cs` file in `AutoLot.Web`:

```
global using AutoLot.Dal.Repos.Base;
global using AutoLot.Dal.Repos.Interfaces.Base;
global using AutoLot.Models.Entities.Base;
global using Microsoft.AspNetCore.Mvc.Rendering;
```

- Create a new folder named `Base` in the `Pages` folder, and in this folder, create a new class named `BasePageModel`. Update the code to the following:

```
namespace AutoLot.Web.Pages.Base;

public abstract class BasePageModel<TEntity>(
  IAppLogging appLoggingInstance,
  IBaseRepo<TEntity> baseRepoInstance,
  string pageTitle) : PageModel where TEntity : BaseEntity, new()
{
  protected readonly IAppLogging AppLoggingInstance = appLoggingInstance;
  protected readonly IBaseRepo<TEntity> BaseRepoInstance = baseRepoInstance;

  [ViewData]
  public string Title { get; init; } = pageTitle;

  [BindProperty]
  public TEntity Entity { get; set; }

  public SelectList LookupValues { get; set; }
  public string Error { get; set; }
  protected virtual void GetLookupValues() => LookupValues = null;
//rest of the code goes here
}
```

Step 2: Add the CRUD methods

- Add the four CRUD methods:

```
public virtual void GetOneEntity(int? id)
{
  if (!id.HasValue)
  {
    Entity = null;
    Error = "Invalid Request";
    return;
  }
  Entity = BaseRepoInstance.Find(id);
  Error = Entity == null ? "Not found" : string.Empty;
}
public virtual IActionResult SaveOne(Func<TEntity, bool, int> saveFunction)
{
  if (!ModelState.IsValid)
  {
    return Page();
  }
  _ = saveFunction(Entity, true);
  return RedirectToPage("Details", new { id = Entity.Id});
}
public virtual IActionResult SaveOneWithLookup(Func<TEntity, bool, int> saveFunction)
{
  if (!ModelState.IsValid)
  {
    GetLookupValues();
    return Page();
  }
  _ = saveFunction(Entity, true);
  return RedirectToPage("Details", new { id = Entity.Id });
}
public virtual IActionResult DeleteOne(int id)
{
  BaseRepoInstance.Delete(Entity);
  return RedirectToPage("Index");
}
```

- Add the following to the `GlobalUsings.cs` file in `AutoLot.Web`:

```
global using AutoLot.Web.Pages.Base;
```

Part 4: Add the Car Templates and List Partial

Step 1: Create the Templates and Partial

- Under the `Cars` folder, create three new folders: `DisplayTemplates`, `EditorTemplates`, and `Partials`.
- Add a new empty Razor view named `Car.cshtml` under the `Pages\Cars\DisplayTemplates` folder. Update the markup to the following:

```
@model 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.Price)</dt>
  <dd class="col-sm-10">@Html.DisplayFor(model => model.Price)</dd>
  <dt class="col-sm-2">@Html.DisplayNameFor(model => model.DateBuilt)</dt>
  <dd class="col-sm-10">@Html.DisplayFor(model => model.DateBuilt)</dd>
  <dt class="col-sm-2">@Html.DisplayNameFor(model => model.IsDrivable)</dt>
  <dd class="col-sm-10">@Html.DisplayFor(model => model.IsDrivable)</dd>
</dl>
```

- Add a new empty Razor view named `CarWithColors.cshtml` under the `Pages\Cars\DisplayTemplates` folder. Update the markup to the following:

```
@model Car
<hr />
<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" style="color:@Model.Color">@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.Price)</dt>
  <dd class="col-sm-10">@Html.DisplayFor(model => model.Price)</dd>
  <dt class="col-sm-2">@Html.DisplayNameFor(model => model.DateBuilt)</dt>
  <dd class="col-sm-10">@Html.DisplayFor(model => model.DateBuilt)</dd>
  <dt class="col-sm-2">@Html.DisplayNameFor(model => model.IsDrivable)</dt>
  <dd class="col-sm-10">@Html.DisplayFor(model => model.IsDrivable)</dd>
</dl>
```

- Add a new empty Razor view named `Car.cshtml` under the `Pages\Cars\EditorTemplates` folder. Update the markup to the following:

```
@model Car
<div asp-validation-summary="All" class="text-danger"></div>
<div>
  <label asp-for="MakeId" class="col-form-label"></label>
  <select asp-for="MakeId" class="form-control" asp-items="@ViewBag.LookupValues"></select>
  <span asp-validation-for="MakeId" class="text-danger"></span>
</div>
<div>
  <label asp-for="Color" class="col-form-label"></label>
  <input asp-for="Color" class="form-control"/>
  <span asp-validation-for="Color" class="text-danger"></span>
</div>
<div>
  <label asp-for="PetName" class="col-form-label"></label>
  <input asp-for="PetName" class="form-control" />
  <span asp-validation-for="PetName" class="text-danger"></span>
</div>
<div>
  <label asp-for="Price" class="col-form-label"></label>
  <input asp-for="Price" class="form-control"/>
  <span asp-validation-for="Price" class="text-danger"></span>
</div>
<div>
  <label asp-for="DateBuilt" class="col-form-label"></label>
  <input asp-for="DateBuilt" class="form-control"/>
  <span asp-validation-for="DateBuilt" class="text-danger"></span>
</div>
<div>
  <label asp-for="IsDrivable" class="col-form-label"></label>
  <input asp-for="IsDrivable" />
  <span asp-validation-for="IsDrivable" class="text-danger"></span>
</div>
```

- Add a new empty Razor view named `_CarList.cshtml` under the `Pages\Cars\Partials` folder. Update the markup to the following:

```
@model IEnumerable<Car>
@{
  var showMake = true;
  if (bool.TryParse(ViewBag.ByMake?.ToString(), out bool byMake))
  {
    showMake = !byMake;
  }
}
<p><item-create></item-create></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.Price)</th>
      <th>@Html.DisplayNameFor(model => model.DateBuilt)</th>
      <th>@Html.DisplayNameFor(model => model.IsDrivable)</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.Price)</td>
      <td>@Html.DisplayFor(modelItem => item.DateBuilt)</td>
      <td>@Html.DisplayFor(modelItem => item.IsDrivable)</td>
      <td>
        <item-edit item-id="@item.Id"></item-edit> |
        <item-details item-id="@item.Id"></item-details> |
        <item-delete item-id="@item.Id"></item-delete>
      </td>
    </tr>
  }
  </tbody>
</table>
```

Step 2: Update the Razor Syntax Page View

- Update the bottom of the `RazorSyntax` view to the following:

```
@* If the templates were in the shared folder, you wouldn't need the full path listed*@
<div class="container">
  @Html.DisplayFor(c=>c.Entity,"Cars/DisplayTemplates/Car.cshtml")
  <hr/>
  @Html.DisplayFor(c=>c.Entity,"Cars/DisplayTemplates/CarWithColors.cshtml")
</div>
Car Editor:
@Html.EditorFor(c=>c.Entity, "Cars/EditorTemplates/Car.cshtml")
<hr/>
<a asp-page="/Cars/Details" asp-route-id="@Model.Entity.Id">@Model.Entity.PetName</a>
```

Part 5: Complete the Car Pages

Copilot Agent Mode

These prompts complete Part 5 of this lab.

```
Prompt: In the Pages/Cars folder, update the IndexModel class to inherit from BasePageModel with
TEntity as Car, ICarRepo, and "Inventory" as pageTitle.
Add a new property named CarRecords of type IList<Car> with get and set accessors. Update the
OnGet method to:
If makeId does not have a value, assign "All Makes" to MakeName CarRecords to the result of
calling GetAllIgnoreQueryFilters on BaseRepoInstance
If makeId does have a value, set MakeName from the parameter and call GetAllBy and assign the
records returned to CarRecords.

Prompt: update the Index page View to the following:
@page "{makeId?}/{makeName?}"
@model AutoLot.Web.Pages.Cars.IndexModel
@{
  if (Model.MakeId.HasValue)
  {
    <h1>Vehicle Inventory for @Model.MakeName</h1>
    var mode = new ViewDataDictionary(ViewData) { { "ByMake", true } };
    <partial name="Partials/_CarList" model="@Model.CarRecords" view-data="@mode" />
  }
  else
  {
    <h1>Vehicle Inventory</h1>
    <partial name="Partials/_CarList" model="@Model.CarRecords" />
  }
}

Prompt: In the Pages/Cars folder, add a new Razor Page named Details that inherits from
BasePageModel with TEntity as Car, ICarRepo, and "Details" as pageTitle.
Update the OnGet method to accept int? id and the following logic:
If id does not have a value, set Entity to null and Error to "Invalid Request".
If it does have a value, assign Entity to the result of calling GetOneEntity with id.
If the result is null, set Error to "Not found", else set Error to string.Empty.
```

```
Prompt: Update the Details page View to the following:
@page "{id?}"
@model AutoLot.Web.Pages.Cars.DetailsModel

<h1>Details for @Model.Entity.PetName</h1>
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert">
    @Model.Error
  </div>
}
else
{
  @Html.DisplayFor(m => m.Entity)
  <hr/>
  @Html.DisplayFor(m => m.Entity, "CarWithColors")
  <div>
    <item-edit item-id="@Model.Entity.Id"></item-edit> |
    <item-delete item-id="@Model.Entity.Id"></item-delete> |
    <item-list></item-list>
  </div>
}

Prompt: In the Pages/Cars folder, add a new Razor Page named Delete that inherits from
BasePageModel with TEntity as Car, ICarRepo, and "Delete" as pageTitle.
Update the OnGet method to accept int? id and the following logic:
If id does not have a value, set Entity to null and Error to "Invalid Request". Then return.
If it does have a value, assign Entity to the result of calling GetOne on the base class with id.
If the result is null, set Error to "Not found", else set Error to string.Empty.
Add an OnPost method to accept int id and the following logic:
If id does not match the Id of the Entity, set Error to "Invalid Request".
Call DeleteOne on the base class, set the Error to string.Empty and Entity to null.
```

```
Prompt: Update the Delete page View to the following:
@page "{id?}"
@model AutoLot.Web.Pages.Cars.DeleteModel

<h1>Delete @Model.Entity.PetName</h1>
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert">
    @Model.Error
  </div>
}
else
{
  <h3>Are you sure you want to delete this car?</h3>
  <div>
    @Html.DisplayFor(c=>c.Entity)
    <form asp-page="Delete" asp-route-id="@Model.Entity.Id">
      <input type="hidden" asp-for="Entity.Id"/>
      <input type="hidden" asp-for="Entity.TimeStamp"/>
      <button type="submit" class="btn btn-danger">
         Delete <i class="fa-solid fa-trash"></i>
      </button>  |  
      <item-list></item-list>
    </form>
  </div>
}

Prompt: In the Pages/Cars folder, add a new Razor Page named Edit that inherits from BasePageModel
with TEntity as Car, ICarRepo, and "Edit" as pageTitle.
Also add in IMakeRepo as a constructor parameter. Do not assign it to a class level variable.
Update the OnGet method to accept int? id and the following logic:
If id does not have a value, set Entity to null and Error to "Invalid Request".
If it does have a value, assign Entity to the result of calling GetOne with id and
GetLookupValues() (both from the base class).
If the result is null, set Error to "Not found", else set Error to string.Empty.
Add an OnPost method to accept int id and the following logic:
If id does not match the Id of the Entity, set Error to "Invalid Request".
Call SaveOneWithLookup(BaseRepoInstance.Update), then set the Error to string.Empty.
Add an override for GetLookupValues that creates a SelectList from calling GetAll on the MakeRepo
instance sorted by Name, Id as the value, Name as the text,
and assign the result it to LookupValues.
```


Prompt: Update the Edit page View to the following:

```
@page "{id?}"
@model AutoLot.Web.Pages.Cars.EditModel
<h1>Edit @Model.Entity.PetName</h1>
<hr/>
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert"> @Model.Error </div>
}
else
{
  <form asp-page="Edit" asp-route-id="@Model.Entity.Id">
    <div class="row">
      <div class="col-md-4">
        <div asp-validation-summary="ModelOnly"></div>
          @Html.EditorFor(x => x.Entity, new { LookupValues = Model.LookupValues })
          <input type="hidden" asp-for="Entity.Id"/>
          <input type="hidden" asp-for="Entity.TimeStamp"/>
        </div>
      </div>
      <div class="d-flex flex-row mt-3">
        <button type="submit" class="btn btn-primary">Save
          <i class="fa-solid fa-save"></i></button>  |  
        <item-list></item-list>
    </div>
  </form>
}
@section Scripts {
  @{ await Html.RenderPartialAsync("_ValidationScriptsPartial"); }
}
```

Prompt: In the Pages/Cars folder, add a new Razor Page named Create that inherits from BasePageModel with TEntity as Car, ICarRepo, and "Create" as pageTitle.
Also add in IMakeRepo as a constructor parameter. Do not assign it to a class level variable.
Update the OnGet method to call GetLookupValues() and set Entity to a new Car instance with IsDrivable set to true.
Add an OnPost method to Call SaveOneWithLookup(BaseRepoInstance.Add), and return the result.
Add an override for GetLookupValues that creates a SelectList from calling GetAll on the MakeRepo instance sorted by Name, nameof(Make.Id) as the value, nameof(Make.Name) as the text, and assign the result it to LookupValues.

```
Prompt: Update the Edit page View to the following:
@page
@model AutoLot.Web.Pages.Cars.CreateModel
<h1>Create a New Car</h1>
<hr />
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert"> @Model.Error </div>
}
else
{
  <form asp-page="Create">
    <div class="row">
      <div class="col-md-4">
         <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            @Html.EditorFor(x => x.Entity, new { LookupValues = Model.LookupValues })
         </div>
      </div>
      <div class="d-flex flex-row mt-3">
        <button type="submit" class="btn btn-success">Create
          <i class="fa-solid fa-plus"></i></button>  |  
        <item-list></item-list>
      </div>
  </form>
  @section Scripts {
    <partial name="_ValidationScriptsPartial" />
  }
}
```

Manual

Step 1: Update the Index Page

- Update the code-behind file to the following:

```
namespace AutoLot.Web.Pages.Cars;
public class IndexModel(IAppLogging appLogging, ICarRepo repo)
  : BasePageModel<Car>(appLogging, repo, "Inventory")
{
  public string MakeName { get; set; }
  public int? MakeId { get; set; }
  public IList<Car> CarRecords { get; set; }
  public void OnGet(int? makeId, string makeName)
  {
    MakeId = makeId;
    if (!makeId.HasValue)
    {
      MakeName = "All Makes";
      CarRecords = carRepo.GetAllIgnoreQueryFilters().ToList();
      return;
    }
    MakeName = makeName;
    CarRecords = carRepo.GetAllBy(makeId.Value).ToList();
  }
}
```

- Update the markup in the View to the following:

```
@page "{makeId?}/{makeName?}"
@model AutoLot.Web.Pages.Cars.IndexModel
@{
  if (Model.MakeId.HasValue)
  {
    <h1>Vehicle Inventory for @Model.MakeName</h1>
    var mode = new ViewDataDictionary(ViewData) { { "ByMake", true } };
    <partial name="Partials/_CarList" model="@Model.CarRecords" view-data="@mode" />
  }
  else
  {
    <h1>Vehicle Inventory</h1>
    <partial name="Partials/_CarList" model="@Model.CarRecords" />
  }
}
```

Step 2: Add the Details Page

- Add a new Razor Page named `Details` to the `Cars` folder. Update the code-behind file to the following:

```
namespace AutoLot.Web.Pages.Cars;

public class DetailsModel(IAppLogging appLogging, ICarRepo repo)
  : BasePageModel<Car>(appLogging, repo, "Details")
{
  if (!id.HasValue)
  {
    Entity = null;
    Error = "Invalid Request";
    return;
  }
  Entity = BaseRepoInstance.Find(id);
  Error = Entity == null ? "Not found" : string.Empty;
}
```

- Update the markup to the following:

```
@page "{id?}"
@model AutoLot.Web.Pages.Cars.DetailsModel

<h1>Details for @Model.Entity?.PetName</h1>
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert">
    @Model.Error
  </div>
}
else
{
  @Html.DisplayFor(m => m.Entity)
  <hr/>
  @Html.DisplayFor(m => m.Entity, "CarWithColors")
  <div>
    <item-edit item-id="@Model.Entity.Id"></item-edit> |
    <item-delete item-id="@Model.Entity.Id"></item-delete> |
    <item-list></item-list>
  </div>
}
```

Step 3: Add the Delete Page

- Add a new Razor Page named `Delete` to the `Cars` folder. Update the code behind file to the following:

```
namespace AutoLot.Web.Pages.Cars;
public class DeleteModel(IAppLogging appLogging, ICarRepo repo)
  : BasePageModel<Car>(appLogging, repo, "Delete")
{
  public void OnGet(int? id)
  {
    if (!id.HasValue)
    {
      Error = "Invalid request";
      Entity = null;
      return;
    }
    GetOne(id);
  }
  public void OnPost(int id)
  {
    if (Entity == null || id != Entity.Id)
    {
      Error = "Invalid Request";
      return;
    }
    DeleteOne(id);
    Error = string.Empty;
    Entity = null;
  }
}
```

- Update the markup to the following:

```
@page "{id?}"
@model AutoLot.Web.Pages.Cars.DeleteModel

<h1>Delete @Model.Entity.PetName</h1>
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert">
    @Model.Error
  </div>
}
else
{
  <h3>Are you sure you want to delete this car?</h3>
  <div>
    @Html.DisplayFor(c=>c.Entity)
    <form asp-page="Delete" asp-route-id="@Model.Entity.Id">
      <input type="hidden" asp-for="Entity.Id"/>
      <input type="hidden" asp-for="Entity.TimeStamp"/>
      <button type="submit" class="btn btn-danger">
         Delete <i class="fa-solid fa-trash"></i>
      </button>  |  
      <item-list></item-list>
    </form>
  </div>
}
```

Step 4: Add the Edit Page

- Add a new Razor Page named `Edit` to the `Cars` folder. Update the code-behind file to the following:

```
namespace AutoLot.Web.Pages.Cars;
public class EditModel(IAppLogging appLogging, ICarRepo carRepo, IMakeRepo makeRepo)
  : BasePageModel<Car>(appLogging, carRepo, "Edit")
{
  public override void GetLookupValues()
  {
    LookupValues = new SelectList(makeRepo.GetAll().OrderBy(m => m.Name).ToList(),
      nameof(Make.Id),
      nameof(Make.Name));
  }
  public void OnGet(int? id)
  {
    if (!id.HasValue)
    {
      Entity = null;
      Error = "Invalid Request";
      return;
    }
    GetOneEntity(id);
    GetLookupValues();
    Error = Entity == null ? "Not found" : string.Empty;
  }
  public void OnPost(int id)
  {
    if (Entity == null || id != Entity.Id)
    {
      Error = "Invalid Request";
      return;
    }
    SaveOneWithLookup(BaseRepoInstance.Update);
    Error = string.Empty;
  }
}
```

- Update the markup to the following:

```
@page "{id?}"
@model AutoLot.Web.Pages.Cars.EditModel
<h1>Edit @Model.Entity.PetName</h1>
<hr/>
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert"> @Model.Error </div>
}
else
{
  <form asp-page="Edit" asp-route-id="@Model.Entity.Id">
    <div class="row">
      <div class="col-md-4">
        <div asp-validation-summary="ModelOnly"></div>
          @Html.EditorFor(x => x.Entity, new { LookupValues = Model.LookupValues })
          <input type="hidden" asp-for="Entity.Id"/>
          <input type="hidden" asp-for="Entity.TimeStamp"/>
        </div>
      </div>
      <div class="d-flex flex-row mt-3">
        <button type="submit" class="btn btn-primary">Save
          <i class="fa-solid fa-save"></i></button>  |  
        <item-list></item-list>
    </div>
  </form>
}
@section Scripts {
  @{ await Html.RenderPartialAsync("_ValidationScriptsPartial"); }
}
```

Step 5: Add the Create Page

- Add a new Razor Page named `Create` to the `Cars` folder. Update the code behind file to the following:

```
namespace AutoLot.Web.Pages.Cars;
public class CreateModel(IAppLogging appLogging,ICarRepo carRepo,IMakeRepo makeRepo)
  : BasePageModel<Car>(appLogging, carRepo, "Create")
{
  public override void GetLookupValues()
  {
    LookupValues = new SelectList(makeRepo.GetAll().OrderBy(m => m.Name).ToList(),
      nameof(Make.Id),
      nameof(Make.Name));
  }
  public void OnGet(int? id)
  {
    if (!id.HasValue)
    {
      Entity = null;
      Error = "Invalid Request";
      return;
    }
    GetOneEntity(id);
    GetLookupValues();
    Error = Entity == null ? "Not found" : string.Empty;
  }
  public void OnPost(int id)
  {
    if (Entity == null || id != Entity.Id)
    {
      Error = "Invalid Request";
      return;
    }
    SaveOneWithLookup(BaseRepoInstance.Add);
    Error = string.Empty;
  }
}
```

- Update the markup to the following:

```
@page
@model AutoLot.Web.Pages.Cars.CreateModel
<h1>Create a New Car</h1>
<hr />
@if (!string.IsNullOrEmpty(Model.Error))
{
  <div class="alert alert-danger" role="alert"> @Model.Error </div>
}
else
{
  <form asp-page="Create">
    <div class="row">
      <div class="col-md-4">
         <div asp-validation-summary="ModelOnly" class="text-danger"></div>
            @Html.EditorFor(x => x.Entity, new { LookupValues = Model.LookupValues })
         </div>
      </div>
      <div class="d-flex flex-row mt-3">
        <button type="submit" class="btn btn-success">Create
          <i class="fa-solid fa-plus"></i></button>  |  
        <item-list></item-list>
      </div>
  </form>
  @section Scripts {
    <partial name="_ValidationScriptsPartial" />
  }
}
```

Summary

In this lab you created the `BasePageModel` and finished the Cars Pages.

Next steps

In the next part of this tutorial series, you will create the custom validation attributes.

