

.NET 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@:&nbsp;
}
<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 `` 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

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;
```

- 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, TPageModel>(
    IAppLogging<TPageModel> appLoggingInstance,
    IBaseRepo<TEntity> baseRepoInstance,
    string pageTitle) : PageModel where TEntity : BaseEntity, new()
{
    protected readonly IAppLogging<TPageModel> 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;
    }
}
```

Step 2: Add the CRUD methods

- Add the four CRUD methods:

```
protected virtual void GetOne(int? id)
{
    if (!id.HasValue)
    {
        Error = "Invalid request";
        Entity = null;
        return;
    }
    Entity = BaseRepoInstance.Find(id.Value);
    if (Entity == null)
    {
        Error = "Not found";
        return;
    }
    Error = string.Empty;
}
```

```

protected virtual IActionResult SaveOne(Func<TEntity,bool,int> saveFunction)
{
    if (!ModelState.IsValid)
    {
        return Page();
    }
    try
    {
        saveFunction(Entity, true);
        return RedirectToPage("./Details", new { id = Entity.Id });
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
        return HandleErrorReturnPage(ex);
    }
}

protected virtual IActionResult SaveWithLookup(Func<TEntity,bool,int> saveFunction)
{
    if (!ModelState.IsValid)
    {
        GetLookupValues();
        return Page();
    }
    try
    {
        saveFunction(Entity, true);
        return RedirectToPage("./Details", new { id = Entity.Id });
    }
    catch (Exception ex)
    {
        ModelState.AddModelError(string.Empty, ex.Message);
        GetLookupValues();
        return HandleErrorReturnPage(ex);
    }
}

protected virtual IActionResult DeleteOne(int id)
{
    try
    {
        BaseRepoInstance.Delete(Entity);
        return RedirectToPage("./Index");
    }
    catch (Exception ex)
    {
        ModelState.Clear();
        Entity = BaseRepoInstance.Find(id);
        return HandleErrorReturnPage(ex);
    }
}

internal IActionResult HandleErrorReturnPage(Exception ex)
{
    Error = ex.Message;
    AppLoggingInstance.LogAppError(ex, "An error occurred");
    return Page();
}

```

- 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:

```
<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

Step 1: Update the Index Page

- Update the code-behind file to the following:

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

- 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<DetailsModel> appLogging, ICarRepo repo)
    : BasePageModel<Car, DetailsModel>(appLogging, repo, "Details")
{
    public void OnGet(int? id) => GetOne(id);
}
```

- 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<DeleteModel> appLogging, ICarRepo repo)
    : BasePageModel<Car, DeleteModel>(appLogging, repo, "Delete")
{
    public void OnGet(int? id)
    {
        if (!id.HasValue)
        {
            Error = "Invalid request";
            Entity = null;
            return;
        }
        GetOne(id);
    }
    public IActionResult OnPost(int id) => DeleteOne(id);
}
```

- Update the markup to the following:

[illegible]

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<EditModel> appLogging, ICarRepo carRepo, IMakeRepo makeRepo)
    : BasePageModel<Car, EditModel>(appLogging, carRepo, "Edit")
{
    public void OnGet(int id)
    {
        GetLookupValues();
        GetOne(id);
    }
    public IActionResult OnPost()
    {
        return SaveWithLookup(BaseRepoInstance.Update);
    }
    protected override void GetLookupValues()
    {
        LookupValues = new SelectList(makeRepo.GetAll(), nameof(Make.Id), nameof(Make.Name));
    }
}
```

- Update the markup to the following:

[illegible]

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<CreateModel> appLogging, ICarRepo carRepo, IMakeRepo makeRepo)
    : BasePageModel<Car, CreateModel>(appLogging, carRepo, "Create")
{
    public void OnGet()
    {
        GetLookupValues();
        Entity = new Car { IsDrivable = true };
    }
    public IActionResult OnPostCreateNewCar() => SaveWithLookup( BaseRepoInstance.Add);
    protected override void GetLookupValues()
    {
        LookupValues = new SelectList(makeRepo.GetAll(), nameof(Make.Id), nameof(Make.Name));
    }
}
```

- Update the markup to the following:

[illegible]

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.