

Build an ASP.NET Core MVC App with EF Core

One-Day Hands-On Lab

Lab 11

This lab walks you through creating a View Component. Prior to starting this lab, you must have completed Lab 14.

Part 1: Adding the Menu View Component

Step 1: Update the Global Usings

- Add the following global using statements to the GlobalUsings.cs file:

```
global using AutoLot.Models.Entities;  
global using Microsoft.AspNetCore.Mvc.ViewComponents;
```

Step 2: Create the View Component Server-Side Code

- Add the following global using statements to the GlobalUsings.cs file:

```
global using AutoLot.Models.Entities;  
global using Microsoft.AspNetCore.Mvc.ViewComponents;
```

- Create a new folder in the MVC project named ViewComponents and add a new class named MenuViewComponent.cs.
- Make the class public and inherit from ViewComponent:

```
namespace AutoLot.Mvc.ViewComponents;  
  
public class MenuViewComponent : ViewComponent  
{  
}
```

- Add a constructor that takes an instance of the IMakeRepo and a private variable to hold the instance.

```
private readonly IMakeRepo _makeRepo;  
public MenuViewComponent(IMakeRepo makeRepo)  
{  
    _makeRepo = makeRepo;  
}
```

- **Note: Only implement the Invoke **or** the InvokeAsync method, not both (or comment one out)**

- Implement the Invoke method (using Make Repository):

```
public IActionResult Invoke()
{
    var makes = _makeRepo.GetAll().ToList();
    if (!makes.Any())
    {
        return new ContentViewComponentResult("Unable to get the makes");
    }
    return View("MenuView", makes);
}
```

- **OR** Implement the Invoke method (using Make Repository):

```
public async Task<IViewComponentResult> InvokeAsync()
{
    return await Task.Run<IViewComponentResult>(() =>
    {
        var makes = _makeRepo.GetAll();
        if (makes == null)
        {
            return new ContentViewComponentResult("Unable to get the makes");
        }
        return View("MenuView", makes);
    });
}
```

Step 3: Update the ViewImports.cshtml File

- To use the ViewComponent as a Tag Helper, the assembly must be registered in the _ViewImports.cshtml file. Add the following to the end of the file:

```
@addTagHelper *, AutoLot.Mvc
```

Step 4: Create the MenuView partial view

- Add a new folder named Components under the Views\Shared folder. Add a new folder named Menu under the Components folder. Add a new partial view named MenuView.cshtml in the new folder.
- Update the code to match the following:

```
@model IEnumerable<Make>
<div class="dropdown-menu">
<a class="dropdown-item text-dark" asp-area="" asp-controller="Cars" asp-action="Index">All</a>

@foreach (var item in Model)
{
    <a class="dropdown-item text-dark" asp-controller="Cars" asp-action="ByMake" asp-route-
makeId="@item.Id" asp-route-makeName="@item.Name">@item.Name</a>
}
</div>
```

Step 5: Update the _Menu.cshtml Partial View

- Open the _Menu.cshtml file in Views\Shared\Partials folder and add the view component as a tag helper before each of the Privacy menu items:

```
<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>
    <vc:menu/>
  </li>
  ...
</ul>
```

Step 6: Stub out the Cars Controller

- Add a new file named CarsController.cs in the Controllers directory. Stub out the following methods on the controller:

```
namespace AutoLot.Mvc.Controllers;

[Route("[controller]/[action]")]
public class CarsController : Controller
{
    [Route("/[controller]")]
    [Route("/[controller]/[action]")]
    [HttpGet]
    public IActionResult Index() => View();
    [HttpGet("{makeId}/{makeName}")]
    public IActionResult ByMake(int makeId, string makeName)
    {
        return View();
    }
    [HttpGet("{id?}")]
    public IActionResult Details(int? id)
    {
        return View();
    }
    [HttpGet]
    public IActionResult Create()
    {
        return View();
    }
    [HttpGet]
    public IActionResult Edit(int? id)
    {
        return View();
    }
    [HttpGet]
    public IActionResult Delete(int? id)
    {
        return View();
    }
}
```

- **Note:** This will be completed in the next lab. The controller class and action methods are needed for the MenuViewComponent and the Tag Helpers.
- If you run the app now, you will see the drop down menu of the makes, although none of the menu items take you to a working page because the views don't exist yet..

Part 2: Adding the Custom Tag Helpers

Step 1: Update the GlobalUsings.cs file

- Add the following to the GlobalUsings.cs file:

```
global using Microsoft.AspNetCore.Mvc.Routing;
global using Microsoft.AspNetCore.Razor.TagHelpers;
global using AutoLot.Mvc.TagHelpers;
global using AutoLot.Mvc.TagHelpers.Base;
```

Step 2: Create the ItemLinkTagHelperBase

- Create a new folder in the MVC project named TagHelpers and add another folder named Base under the TagHelpers folder. In the Base folder, add a new class named ItemLinkTagHelperBase.cs. Make the class public and abstract, and inherit from TagHelper:

```
namespace AutoLot.Mvc.TagHelpers.Base;

public abstract class ItemLinkTagHelperBase : TagHelper
{
}
```

- Add a protected constructor that accepts an instance of IActionContextAccessor and IUrlHelperFactory. Use them to create an instance of IUrlHelper. Use the IActionContextAccessor to get the current controller name from the route values:

```
protected readonly IUrlHelper UrlHelper;
private string _controllerName
protected ItemLinkTagHelperBase(
    IActionContextAccessor contextAccessor, IUrlHelperFactory urlHelperFactory)
{
    UrlHelper = urlHelperFactory.GetUrlHelper(contextAccessor.ActionContext);
    _controllerName = contextAccessor.ActionContext.ActionDescriptor.RouteValues["controller"];
}
```

- Add a public property to hold the item id. The protected property becomes part of the tag helper signature:

```
public int? ItemId { get; set; }
```

- Add a protected property to hold the action name:

```
protected string ActionName { get; set; }
```

- Implement the protected BuildContent method:

```
protected void BuildContent(TagHelperOutput output,
    string cssClassName, string displayText, string fontAwesomeName)
{
    output.TagName = "a"; // Replaces <email> with <a> tag
    var target = (ItemId.HasValue)
        ? UrlHelper.Action(ActionName, controllerName, new {id = ItemId})
        : UrlHelper.Action(ActionName, controllerName);
    output.Attributes.SetAttribute("href", target);
    output.Attributes.Add("class", cssClassName);
    output.Content.AppendHtml($"@{{displayText}} <i class=""fas fa-{{fontAwesomeName}}""></i>");
}
```

Step 3: Create the ItemCreateTagHelper

- In the TagHelpers folder, add a new class named ItemCreateTagHelper.cs and make the class public and inherit from ItemLinkTagHelperBase:

```
namespace AutoLot.Mvc.TagHelpers;

public class ItemCreateTagHelper : ItemLinkTagHelperBase
{
    public ItemCreateTagHelper(
        IActionContextAccessor contextAccessor,
        IUrlHelperFactory urlHelperFactory)
        : base(contextAccessor, urlHelperFactory)
    {
        ActionName = nameof(CarsController.Create);
    }
}
```

- Override Process and call into the base BuildContent method:

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
    BuildContent(output, "text-success", "Create New", "plus");
}
```

Step 4: Create the ItemDeleteTagHelper

- In the TagHelpers folder, add a new class named ItemDeleteTagHelper.cs and make the class public and inherit from ItemLinkTagHelperBase:

```
namespace AutoLot.Mvc.TagHelpers;

public class ItemDeleteTagHelper : ItemLinkTagHelperBase
{
    public ItemDeleteTagHelper(
        IActionContextAccessor contextAccessor,
        IUrlHelperFactory urlHelperFactory)
        : base(contextAccessor, urlHelperFactory)
    {
        ActionName = nameof(CarsController.Delete);
    }
}
```

- Override Process and call into the base BuildContent method:

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

Step 5: Create the ItemDetailsTagHelper

- In the TagHelpers folder, add a new class named ItemDetailsTagHelper.cs and make the class public and inherit from ItemLinkTagHelperBase:

```
namespace AutoLot.Mvc.TagHelpers;

public class ItemDetailsTagHelper : ItemLinkTagHelperBase
{
    public ItemDetailsTagHelper(
        IActionContextAccessor contextAccessor,
        IUrlHelperFactory urlHelperFactory)
        : base(contextAccessor, urlHelperFactory)
    {
        ActionName = nameof(CarsController.Details);
    }
}
```

- Override Process and call into the base BuildContent method:

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

Step 6: Create the ItemEditTagHelper

- In the TagHelpers folder, add a new class named ItemEditTagHelper.cs and make the class public and inherit from ItemLinkTagHelperBase:

```
namespace AutoLot.Mvc.TagHelpers;

public class ItemEditTagHelper : ItemLinkTagHelperBase
{
    public ItemEditTagHelper(
        IActionContextAccessor contextAccessor,
        IUrlHelperFactory urlHelperFactory)
        : base(contextAccessor, urlHelperFactory)
    {
        ActionName = nameof(CarsController.Edit);
    }
}
```

- Override Process and call into the base BuildContent method:

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

Step 7: Create the List Items TagHelper

- In the TagHelpers folder, add a new class named `ItemListTagHelper.cs` and make the class public and inherit from `ItemLinkTagHelperBase`:

```
namespace AutoLot.Mvc.TagHelpers;

public class ItemListTagHelper : ItemLinkTagHelperBase
{
    public ItemListTagHelper(
        IActionContextAccessor contextAccessor, IUrlHelperFactory urlHelperFactory)
        : base(contextAccessor, urlHelperFactory)
    {
        ActionName = nameof(CarsController.Index);
    }
}
```

- Override `Process` and call into the base `BuildContent` method:

```
public override void Process(TagHelperContext context, TagHelperOutput output)
{
    BuildContent(output, "text-default", "Back to List", "list");
}
```

Summary

The lab created the Menu view component and the custom tag helpers.

Next steps

In the next part of this tutorial series, you will build the `BaseCrudController` and complete the `CarsController`.