.NET 10 App Dev Hands-On Workshop

Blazor Lab 5 –Pages, Navigation, and Validation

This lab adds the application pages and navigation to the `AutoLot.Blazor` project. Before starting this lab, you must have completed Blazor Lab 4.

Part 1: Update the Main Layout and the Home Page

Copilot Agent Mode

```
Prompt: All work is to be done in the AutoLot.Blazor project.

Prompt: Add the following to the _Imports.razor file (don't remove any existing entries):
@using AutoLot.Blazor.Models.Validation
@using Microsoft.AspNetCore.Components
@using Microsoft.AspNetCore.Components.Sections
@using Microsoft.Extensions.Options

Prompt: Remove the About menu item from MainLayout.razor.

Prompt: Add a sectionoutlet to MainLayout.razor with the sectionname of "top-bar" just above the
<article class="content px-4"> entry.

Prompt: Clear out the Home.razor file and replace it with the following content:
@page "/"
@page "/home"
@inject IOptionsSnapshot<DealerInfo> DealerOptionsSnapshot
<h3 class="text-center">@DealerOptionsSnapshot.Value.City, @DealerOptionsSnapshot.Value.State</h3>
<SectionContent SectionName="top-bar">
  <h2 class="text-center">@DealerOptionsSnapshot.Value.DealerName</h2>
</SectionContent>

Prompt: Replace the "AutoLot.Blazor" text in the NavMenu.razor component with the following
"@DealerOptionsSnapshot.Value.DealerName" and inject the IOptionsSnapshot<DealerInfo> service at
the top of the file. The navbar-brand should word wrap.
```

Manual

- Add the following to the _Imports.razor file:

```
@using Microsoft.AspNetCore.Components
@using Microsoft.AspNetCore.Components.Sections
@using Microsoft.Extensions.Options
```

- Remove the following markup from the MainLayout.razor component and replace it with the SectionOutlet:

```
<div class="top-row px-4">
    <a href="https://learn.microsoft.com/aspnet/core/" target="_blank">About</a>
</div>
<SectionOutlet SectionName="top-bar" />
<article class="content px-4">
  @Body
</article>
```

- Clear out the Home.razor page and update the file to have two @page directives and add in the DealerInfo options monitor:

```
@page "/"
@page "/home"
@inject IOptionsSnapshot<DealerInfo> DealerOptionsSnapshot
<h3 class="text-center">@DealerOptionsShapshot.Value.City, @DealerOptionsSnapshot.Value.State</h3>
<SectionContent SectionName="top-bar">
  <h2 class="text-center">@DealerOptionsSnapshot.Value.DealerName</h2>
</SectionContent>
```

- Inject the DealerInfo options snapshot into the NavMenu.razor component and use the dealer name to replace the "AutoLot.Blazor" text in the navbar-brand tag and update it for text wrapping:

```
@inject IOptionsSnapshot<DealerInfo> DealerOptionsSnapshot
<a class="navbar-brand text-wrap" href="">@DealerOptionsShanpshot.Value.DealerName</a>
```

Part 2: Add the Razor Syntax Page

- Create a new Razor component named `RazorSyntax` in the `Pages` folder. Update the code to the following:

```
@page "/razor-syntax"
<PageTitle>Razor Syntax</PageTitle>
<title>Razor Syntax</title>
<h3>Razor Syntax</h3>
@for (int i = 0; i < 15; i++)
{
  @:Counter: @i<br/>
}
@{
  //Code Block
  var foo = "Foo";
  var bar = "Bar";
  var htmlString = "<ul><li>one</li><li>two</li></ul>";
}
@foo<br />
@htmlString<br />
@((MarkupString)htmlString)<br />
@foo.@bar<br />
@foo.ToUpper()<br/>
<hr />
@{
  @:Straight Text
  <div>Value:@_entity.Id</div>
  <text>
    Lines without HTML tag
  </text>
  <br />
}
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 />
<hr />
The Car named @_entity.PetName is a <span style="color:@_entity.Color">@_entity.Color</span>
@_entity.MakeNavigation.Name
<hr />
@code {
  private readonly Car _entity = new Car
  {
    Id = 4, Color = "Yellow", PetName = "Hank", MakeId = 1, IsDrivable = true,
      DateBuilt = new DateTime(2022,01,01),Price="$100,099.00",
    MakeNavigation = new Make {Id = 1, Name = "BMW"}
  };
}
```

- Add the following to the NavMenu.Razor component:

```
<div class="nav-item px-3">
  <NavLink class="nav-link" href="/razor-syntax" Match="NavLinkMatch.All">
    <span class="fa-solid fa-cut pe-2" aria-hidden="true"></span>Razor Syntax
  </NavLink>
</div>
```

Part 3: Add the Privacy Page

- Add a new Razor component named `Privacy.razor` in the `Pages` folder and update the markup to the following:

```
@page "/privacy"
@page "/privacy/{RouteParameter}"
<PageTitle>Privacy Policy</PageTitle>
<title>Privacy Policy</title>
<p>Use this page to detail your site's privacy policy.</p>
@if (!string.IsNullOrEmpty(RouteParameter))
{
  <h3>Route Parameter: @RouteParameter</h3>
}
@if (!string.IsNullOrEmpty(QueryStringParameter))
{
  <h3>Query String Parameter: @QueryStringParameter</h3>
}
```

- Add a new Class file named `Privacy.razor.cs` in the `Pages` folder and update the code to the following:

```
using Microsoft.AspNetCore.Components;
namespace AutoLot.Blazor.Pages;
public partial class Privacy
{
  [Parameter]
  public string RouteParameter { get; set; }
  [Parameter]
  [SupplyParameterFromQuery(Name = "QueryStringParam")]
  public string QueryStringParameter { get; set; }
}
```

- Add the following to the `NavMenu.Razor` component:

```
<div class="nav-item px-3">
  <NavLink class="nav-link"
    href="/privacy/myRouteParameter?QueryStringParam=sent+in+on+query+string"
    Match="NavLinkMatch.All">
    <span class="fa-solid fa-user-secret pe-2" aria-hidden="true"></span>Privacy
  </NavLink>
</div>
```

Part 4: Prepare for Validation

Copilot Agent Mode

Prompt: Add the following global usings to the GlobalUsings.cs file if it does not already exist (sorted alphabetically. Don't remove any existing global using statements).

```
global using Microsoft.Extensions.Validation;
global using Microsoft.AspNetCore.Components.Forms;
```

Prompt: Update the Program.cs file by adding in the following two lines (and not removing any existing lines):

```
builder.Services.AddModelValidation();
builder.Services.AddValidation();
```

Prompt: Add the following to the NavMenu.razor component markup:

```
<div class="nav-item px-3">
  <NavLink class="nav-link" @onclick="() => _expandValidationSubNav = !_expandValidationSubNav">
    <span class="fa-solid fa-check pe-2" aria-hidden="true"></span>
    Validation
    <span class="fa-solid fa-sort-down ps-1" aria-hidden="true"
hidden="@(_expandValidationSubNav)"></span>
    <span class="fa-solid fa-sort-up ps-1" aria-hidden="true"
hidden="@(!_expandValidationSubNav)"></span>
  </NavLink>
  @if (_expandValidationSubNav)
  {
    <NavLink class="nav-link ps-5" href="/validations/shopping-cart" Match="NavLinkMatch.All">
      <span class="fa-solid fa-cart-shopping pe-2" aria-hidden="true"></span>
      Shopping Cart
    </NavLink>
    <NavLink class="nav-link ps-5" href="/validations/car-validation" Match="NavLinkMatch.All">
      <span class="fa-solid fa-car pe-2" aria-hidden="true"></span>
      Car
    </NavLink>
  }
</div>
```

Add the following to the NavMenu.razor @code block:

```
private bool _expandValidationSubNav;
```

Prompt: Create a new folder named Validation in the Services folder, and in that folder create class named CustomFieldClassProvider that inherits from FieldCssClassProvider. Override the GetFieldCssClass method to return "validField" if the field is valid and "invalidField" if the field is invalid.

Prompt: Add the following CSS classes to app.css:

```
.invalidField {
    outline: 1px solid red;
}
.validField {
    outline: 1px solid green;
}
```

Add the following global usings to the GlobalUsings.cs file if it does not already exist (sorted alphabetically. Don't remove any existing global using statements).

```
global using AutoLot.Blazor.Services.Validation;
```

Manual

Step 1: Update the GlobalUsings.cs file

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

```
global using Microsoft.Extensions.Validation;
global using Microsoft.AspNetCore.Components.Forms;
```

Step 2: Update the Program.cs file

- Add the following to the Program.cs file in the AutoLot.Blazor project:

```
builder.Services.AddModelValidation();
builder.Services.AddValidation();
```

Step 3: Add the Validation Menu Items

- Add the validation menus to the `NavMenu.razor` component:

```
<div class="nav-item px-3">
  <NavLink class="nav-link" @onclick="() => _expandValidationSubNav = !_expandValidationSubNav">
    <span class="fa-solid fa-check pe-2" aria-hidden="true"></span>
    Validation
    <span class="fa-solid fa-sort-down ps-1" aria-hidden="true"
hidden="@(_expandValidationSubNav)"></span>
    <span class="fa-solid fa-sort-up ps-1" aria-hidden="true"
hidden="@(!_expandValidationSubNav)"></span>
  </NavLink>
  @if (_expandValidationSubNav)
  {
    <NavLink class="nav-link ps-5" href="/validations/shopping-cart" Match="NavLinkMatch.All">
      <span class="fa-solid fa-cart-shopping pe-2" aria-hidden="true"></span>
      Shopping Cart
    </NavLink>
    <NavLink class="nav-link ps-5" href="/validations/car-validation" Match="NavLinkMatch.All">
      <span class="fa-solid fa-car pe-2" aria-hidden="true"></span>
      Car
    </NavLink>
  }
</div>
```

- Add the following to the `@code` block:

```
private bool _expandValidationSubNav;
```

Step 4: Add the Field Class Provider

- Create a class in the Services folder named `CustomFieldClassProvider.cs` and update it to the following:

```
namespace AutoLot.Blazor.Models.Validation;
public class CustomFieldClassProvider : FieldCssClassProvider
{
  public override string GetFieldCssClass(
    EditContext editContext, in FieldIdentifier fieldIdentifier)
    => editContext.IsValid(fieldIdentifier) ? "validField" : "invalidField";
}
```

- Add the following CSS classes to `app.css`:

```
.invalidField {
    outline: 1px solid red;
}
.validField {
    outline: 1px solid green;
}
```

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

```
global using AutoLot.Blazor.Services.Validation;
```

Part 5: Add the Validation Examples

Step 1: Add the Confirmation Dialog Component

- Add a new Razor component named `ConfirmDialog.razor` in the `Shared` folder and update the code to the following:

```
@if (Show)
{
  <div class="p-3 mt-4" style="border:5px solid red">
    <div>
      <div>@ChildContent</div>
      <div>
        <button @onclick="OnOk">OK</button>
      </div>
    </div>
  </div>
}
@code {
  [Parameter] [EditorRequired] public bool Show { get; set; }
  [Parameter] [EditorRequired] public EventCallback OnOk { get; set; }
  [Parameter] [EditorRequired] public RenderFragment ChildContent { get; set; }
}
```

Step 2: Add the Shopping Cart Validation Page

- Add a new folder named `Validation` in the `Pages` folder, and in that folder, create a new Razor component named `ShoppingCartValidation.razor`. Update the code to the following:

```
@page "/validations/shopping-cart"
@implements IDisposable
<PageTitle>Shopping Cart Validation</PageTitle>
<h3>Shopping Cart Validation</h3>
<div class="row">
  <EditForm EditContext="@editContext" OnValidSubmit="ProcessOrder" OnInvalidSubmit="StopOrder">
    <DataAnnotationsValidator/>
    <ValidationSummary Model="_entity"/>
    <div>
      <label class="col-form-label" for="@nameof(AddToCartViewModel.Id)">Id</label>
      <InputNumber id="@nameof(AddToCartViewModel.Id)" class="form-control"
        @bind-Value="_entity.Id"/>
      <ValidationMessage For="() => _entity.Id"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(AddToCartViewModel.StockQuantity)">
        Stock Quantity
      </label>
      <InputNumber id="@nameof(AddToCartViewModel.StockQuantity)" class="form-control"
        @bind-Value="_entity.StockQuantity"/>
      <ValidationMessage For="() => _entity.StockQuantity"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(AddToCartViewModel.ItemId)">ItemId</label>
      <InputNumber id="@nameof(AddToCartViewModel.ItemId)" class="form-control"
        @bind-Value="_entity.ItemId"/>
      <ValidationMessage For="() => _entity.ItemId"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(AddToCartViewModel.Quantity)">Quantity</label>
      <InputNumber id="@nameof(AddToCartViewModel.Quantity)" class="form-control"
        @bind-Value="_entity.Quantity"/>
      <ValidationMessage For="() => _entity.Quantity"/>
    </div>
    <button class="mt-3" type="submit" disabled="@buttonOneDisabled">
      @buttonOneLabel
    </button>
    <button class="mt-3" type="submit">Process Order 2</button>
  </EditForm>
  <div class="mt-3 @messageClass">@message</div>
</div>
```

```
@code {
  private const string NormalButtonLabel = "Process Order 1";
  private const string processingButtonLabel = "Processing...";
  private bool buttonOneDisabled = true;
  private bool formInvalid = true;
  EditContext editContext;
  private AddToCartViewModel _entity;
  private string message = "";
  private string messageClass = "";
  private string buttonOneLabel = NormalButtonLabel;
  protected override void OnInitialized()
  {
    _entity = new AddToCartViewModel();
    editContext = new EditContext(_entity);
    editContext.SetFieldCssClassProvider(new CustomFieldClassProvider());
    editContext.OnFieldChanged += HandleFieldChanged;
  }
  private bool _hasValidated = false;
  protected override void OnAfterRender(bool firstRender)
  {
    if (firstRender && !_hasValidated)
    {
      formInvalid = !editContext.Validate();
      buttonOneDisabled = formInvalid;
      _hasValidated = true;
      StateHasChanged();
    }
  }
  private void HandleFieldChanged(object sender, FieldChangedEventArgs e)
  {
    if (editContext is null)
    {
      return;
    }
    formInvalid = !editContext.Validate();
    buttonOneDisabled = formInvalid;
  }
  public void Dispose()
  {
    if (editContext is not null)
    {
      editContext.OnFieldChanged -= HandleFieldChanged;
    }
  }
```

```
  public async Task ProcessOrder()
  {
    buttonOneDisabled = true;
    buttonOneLabel = ProcessingButtonLabel;
    await Task.Delay(5000);
    buttonOneLabel = NormalButtonLabel;
    message = "Order Processed";
    messageClass = "alert alert-success";
    buttonOneDisabled = false;
  }
  public void StopOrder()
  {
    message = "Order Stopped";
    messageClass = "alert alert-danger";
  }
}
```

Step 3: Add the Car Validation Page

- Create a new Razor component named `CarValidation.razor` in the `Validations` folder and update the code to the following:

```
@page "/validations/car-validation"
<PageTitle>Car Validation</PageTitle>
<h3>Car Validation</h3>
<div class="row">
  <EditForm Model="_entity" OnValidSubmit="ProcessOrder" OnInvalidSubmit="StopOrder">
    <DataAnnotationsValidator/>
    <ValidationSummary/>
    <div>
      <label class="col-form-label" for="@nameof(Car.Id)">Id</label>
      <InputNumber Id="@nameof(Car.Id)" class="form-control" @bind-Value="_entity.Id"
        DisplayName="Vehicle ID" ParsingErrorMessage="The {0} is Required"/>
      <ValidationMessage For="() => _entity.Id"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(Car.PetName)">Pet Name</label>
      <InputText Id="@nameof(Car.PetName)" class="form-control" @bind-Value="_entity.PetName"/>
      <ValidationMessage For="() => _entity.PetName"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(Car.MakeId)">Make</label>
      <InputSelect Id="@nameof(Car.MakeId)" class="form-control" @bind-Value="_entity.MakeId">
      @foreach (var item in _makes)
      {
        <option value="@item.Id">@item.Name</option>
      }
      </InputSelect>
      <ValidationMessage For="() => _entity.MakeId"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(Car.IsDrivable)">IsDrivable</label>
      <InputCheckbox Id="@nameof(Car.IsDrivable)" @bind-Value="_entity.IsDrivable"/>
      <ValidationMessage For="() => _entity.IsDrivable"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(Car.DateBuilt)">Date Built</label>
      <InputDate Id="@nameof(Car.DateBuilt)" class="form-control"
          @bind-Value="_entity.DateBuilt"/>
      <ValidationMessage For="() => _entity.DateBuilt"/>
    </div>
    <div>
      <label class="col-form-label" for="@nameof(Car.Price)">Price</label>
      <InputText Id="@nameof(Car.Price)" class="form-control" @bind-Value="_entity.Price"/>
      <ValidationMessage For="() => _entity.Price"/>
    </div>
    <div class="pt-4">
      <button>Process Car</button>
    </div>
  </EditForm>
```

```
<ConfirmDialog Show="_showAlert" OnOk="@(() => _showAlert = false)">
  <ChildContent>
    <h1>This will save the content</h1>
    <p>Click OK when ready.</p>
  </ChildContent>
</ConfirmDialog>
</div>

@code {
  private bool _showAlert = false;
  private Car _entity = new Car
  {
    Id = 4, Color = "Yellow", PetName = "Hank", MakeId = 5, IsDrivable = true,
    DateBuilt = new DateTime(2022, 01, 01), Price = "$99,999.99"
  };
  private List<Make> _makes =>
  [
    new() { Id = 1, Name = "VW" },
    new() { Id = 2, Name = "Ford" },
    new() { Id = 3, Name = "Saab" },
    new() { Id = 4, Name = "Yugo" },
    new() { Id = 5, Name = "BMW" },
    new() { Id = 6, Name = "Pinto" }
  ];
  public void ProcessOrder(EditContext context)
  {
    Console.WriteLine($"$Car is valid: {context.Validate()}");
    _showAlert = true;
  }
  public void StopOrder(EditContext context)
  {
    Console.WriteLine($"Car is invalid {string.Join(",", context.GetValidationMessages())}");
  }
}
```

Part 6: Add the Inventory SubMenu

Step 1: Create the Makes Submenu Component

- Add a new Razor component named `MakesSubMenu.razor` to the `Layout` folder. Clear out the contents and update it to the following:

```razor
@if (!_makes.Any())
{
  <div class="text-light px-3">
    <span class="fa-solid fa-spinner pe-2" aria-hidden="true"></span>Loading...
  </div>
}
else
{
  <NavLink class="nav-link ps-5" href="cars/index" Match="NavLinkMatch.All">
    <span aria-hidden="true"></span> All
  </NavLink>
  foreach (var m in _makes)
  {
    var link = $"cars/index/{m.Id}/{m.Name}";
    <NavLink class="nav-link ps-5" href="@link" Match="NavLinkMatch.All">
      <span aria-hidden="true"></span> @m.Name
    </NavLink>
  }
}
@code {
  private List<Make> _makes = [];
  [Inject] private IMakeDataService MakeDataService { get; set; }
  protected override async Task OnInitializedAsync()
  {
    _makes = (await MakeDataService.GetAllEntitiesAsync()).ToList();
  }
}
```

Step 2: Update the NavMenu Component

- Add the following menu item after the Home page menu item:

```
<div class="nav-item px-3">
  <NavLink class="nav-link" @onclick="() => _expandInventorySubNav = !_expandInventorySubNav">
    <span class="fa-solid fa-car pe-2" aria-hidden="true"></span>Inventory
    <span class="fa-solid fa-sort-down ps-1" aria-hidden="true"
hidden="@(_expandInventorySubNav)"></span>
    <span class="fa-solid fa-sort-up ps-1" aria-hidden="true"
hidden="@(!_expandInventorySubNav)"></span>
  </NavLink>
  @if (_expandInventorySubNav)
  {
    <MakesSubMenu></MakesSubMenu>
  }
</div>
```

- Add the following to the @code block:

```
private bool _expandInventorySubNav;
```

Summary

This lab added navigation and some example pages to the client application.

Next Steps

The following lab will build the helpers used by the AutoLot Pages and components.