.NET App Dev Hands-On Workshop

Blazor Lab 5 –Pages, Navigation, and Validation

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

Part 1: Update the MainLayout and the Menus

Step 1: Update the Layout

- Add the following to the `_Imports.razor` file:

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

- Update the `MainLayout.razor` Blazor page (new line in bold):

```
@inherits LayoutComponentBase
<div class="page">
  <div class="sidebar">
    <NavMenu />
  </div>
  <main>
    <SectionOutlet SectionName="top-bar" />
    <article class="content px-4">
      @Body
    </article>
  </main>
</div>
```

Step 2: Add 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:

```
<NavLink class="nav-link ps-5" href="cars/index" Match="NavLinkMatch.All">
  <span aria-hidden="true"></span> All
</NavLink>
@if (!_makes.Any()) { <div><em>Loading...</em></div> }
else
{
  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();
  }
}
```

Step 3: Update the NavMenu Component

- Clear out the contents of the `NavMenu.razor` component and update it to the following:
  NOTE: Most pages won't work if you run the app now.

```
<div class="top-row ps-3 navbar navbar-dark">
  <div class="container-fluid">
    <a class="navbar-brand" href="/home">Skimedic's Used Cars</a>
    <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
      <span class="navbar-toggler-icon"></span>
    </button>
  </div>
</div>
<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
  <nav class="nav flex-column">
    <li class="nav-item px-3">
      <NavLink class="nav-link" href="/home">Home <span class="fa-solid fa-home ps-2"></span>
      </NavLink>
      <NavLink class="nav-link" @onclick="() => _expandInventorySubNav = !_expandInventorySubNav">
        Inventory
        <span class="fa-solid fa-car ps-1 pe-2" aria-hidden="true"></span>
        <span class="fa-solid fa-sort-down pe-2" aria-hidden="true"></span>
      </NavLink>
      @if (_expandInventorySubNav) { <MakesSubMenu></MakesSubMenu> }
      <NavLink class="nav-link" href="blazor-life-cycle">
        LifeCycle <span class="fa-solid fa-bolt-lightning ps-2" aria-hidden="true"></span>
      </NavLink>
      <NavLink class="nav-link" href="/razor-syntax">
        Razor Syntax <span class="fa-solid fa-cut ps-2" aria-hidden="true"></span>
      </NavLink>
      <NavLink class="nav-link" href="/privacy">
        Privacy <span class="fa-solid fa-user-secret ps-2" aria-hidden="true"></span>
      </NavLink>
      <NavLink class="nav-link" href="/validation">
        Validation <span class="fa-solid fa-check ps-2" aria-hidden="true"></span>
      </NavLink>
      <NavLink class="nav-link" href="/car-validation">
        Car Validation <span class="fa-solid fa-car ps-2" aria-hidden="true"></span>
                       <span class="fa fa-check ps-2" aria-hidden="true"></span>
      </NavLink>
      <NavLink class="nav-link" href="/make-validation">
        Make Validation <span class="fa-solid fa-copyright ps-2" aria-hidden="true"></span>
                        <span class="fa-solid fa-check ps-2" aria-hidden="true"></span>
      </NavLink>
    </li>
  </nav>
</div>
```

```
@code {
  private bool _collapseNavMenu = true;
  private string NavMenuCssClass => _collapseNavMenu ? "collapse" : null;
  private bool _expandInventorySubNav;
  private void ToggleNavMenu()
  {
    _collapseNavMenu = !_collapseNavMenu;
  }
}
```

Step 4: Update the Home.razor Page

- 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"
@using Microsoft.Extensions.Options
@inject IOptionsMonitor<DealerInfo> DealerOptionsMonitor
<h3 class="text-center">@DealerOptionsMonitor.CurrentValue.City,
@DealerOptionsMonitor.CurrentValue.State</h3>
<SectionContent SectionName="top-bar">
  <h2 class="text-center">@DealerOptionsMonitor.CurrentValue.DealerName</h2>
</SectionContent>
```

Part 2: Add the Privacy Page

- Add a new Razor component named `Privacy.razor` in the `Pages` folder and update the markup and code 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>
}
@code {
  [Parameter]
  public string RouteParameter { get; set; }

  [Parameter]
  [SupplyParameterFromQuery(Name = "QueryStringParam")]
  public string QueryStringParameter { get; set; }
}
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Part 3: 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"}
  };
}
```

Part 4: 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; }
}
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Part 5: Add the Validation Examples

Step 1: Add the Validation Page

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

```
@page "/validation"
@implements IDisposable
<PageTitle>Validation</PageTitle>
<h3>Validation</h3>
<div class="row">
  <EditForm EditContext="@editContext" OnValidSubmit="ProcessOrder" OnInvalidSubmit="StopOrder">
    <DataAnnotationsValidator />
    <ValidationSummary Model="_entity" />
    <div>
      <label class="col-form-label" for="id">Id</label>
      <InputNumber class="form-control" @bind-Value="_entity.Id" />
      <ValidationMessage For="()=>_entity.Id" />
    </div>
    <div>
      <label class="col-form-label" for="stockQuantity">Stock Quantity</label>
      <InputNumber class="form-control" @bind-Value="_entity.StockQuantity" />
      <ValidationMessage For="()=>_entity.StockQuantity" />
    </div>
    <div>
      <label class="col-form-label" for="itemId">ItemId</label>
      <InputNumber class="form-control" @bind-Value="_entity.ItemId" />
      <ValidationMessage For="()=>_entity.ItemId" />
    </div>
    <div>
      <label class="col-form-label" for="quantity">Quantity</label>
      <InputNumber class="form-control" @bind-Value="_entity.Quantity" />
      <ValidationMessage For="()=>_entity.Quantity" />
    </div>
    <button class="mt-3" type="submit" disabled="@formInvalid">Process Order 1</button>
    <button class="mt-3" type="submit">Process Order 2</button>
  </EditForm>
  <div class="mt-3 @messageClass">@message</div>
</div>
```

```
@code {
  private bool formInvalid = true;
  EditContext editContext;
  private AddToCartViewModel _entity;
  private string message = "";
  private string messageClass = "";
  protected override void OnInitialized()
  {
    _entity = new AddToCartViewModel();
    editContext = new EditContext(_entity);
    editContext.OnFieldChanged += HandleFieldChanged;
  }
  private void HandleFieldChanged(object sender, FieldChangedEventArgs e)
  {
    if (editContext is null)
    {
      return;
    }
    formInvalid = !editContext.Validate();
    StateHasChanged();
  }
  public void Dispose()
  {
    if (editContext is not null)
    {
      editContext.OnFieldChanged -= HandleFieldChanged;
    }
  }
  public void ProcessOrder()
  {
    message = "Order Processed";
    messageClass = "alert alert-success";
  }
  public void StopOrder()
  {
    message = "Order Stopped";
    messageClass = "alert alert-danger";
  }
}
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 2: Add the Make Validation Page

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

```
@page "/make-validation"
<PageTitle>Make Validation</PageTitle>
<h3>Make Validation</h3>
<div class="row">
  <EditForm  Model="_makeEntity" OnValidSubmit="ProcessOrder" OnInvalidSubmit="StopOrder">
    <DataAnnotationsValidator />
    <ValidationSummary />
    <div>
      <label class="col-form-label" for="id">Id</label>
      <InputNumber class="form-control" @bind-Value="_makeEntity.Id" />
      <ValidationMessage For="()=>_makeEntity.Id" />
    </div>
    <div>
      <label class="col-form-label" for="name">Make Name</label>
      <InputText class="form-control" @bind-Value="_makeEntity.Name" />
      <ValidationMessage For="()=>_makeEntity.Name" />
    </div>
    <div class="pt-4">
      <button>Process Make</button>
    </div>
  </EditForm>
</div>

@code {
  private Make _makeEntity = new Make { Id = 1, Name = "VW" };
  public void ProcessOrder(EditContext context)
  {
    Console.WriteLine($"Make is valid: {context.Validate()}");
  }
  public void StopOrder(EditContext context)
  {
    Console.WriteLine($"Make is invalid {string.Join(",",context.GetValidationMessages())}");
  }
}
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 3: Add the Car Validation Page

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

```
@page "/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="id">Id</label>
      <InputNumber class="form-control" @bind-Value="_entity.Id" />
      <ValidationMessage For="()=>_entity.Id" />
    </div>
    <div>
      <label class="col-form-label" for="petName">Pet Name</label>
      <InputText class="form-control" @bind-Value="_entity.PetName" />
      <ValidationMessage For="()=>_entity.PetName" />
    </div>
    <div>
      <label class="col-form-label" for="name">Make Name</label>
      <InputText class="form-control" @bind-Value="_entity.MakeNavigation.Name" />
      <ValidationMessage For="()=>_entity.MakeNavigation.Name" />
    </div>
    <div>
      <label class="col-form-label" for="quantity">Make</label>
      <InputSelect 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 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 = 1, IsDrivable = true,
      DateBuilt = new DateTime(2022, 01, 01), Price = "$100,099.00",
      MakeNavigation = new Make { Id = 1, Name = "BMW" }
    };
  private Make _makeEntity = new Make { Id = 1, Name = "VW" };
  private List<Make> _makes =>
  [
    new() { Id = 1, Name = "VW" },
    new() { Id = 2, Name = "Ford" },
    new() { Id = 3, Name = "Saab" },
    new() { Id = 4, Name = "Yugo" },
    new() { Id = 5, Name = "BMW" },
    new() { Id = 6, Name = "Pinto" }
  ];
  public void ProcessOrder(EditContext context)
  {
    Console.WriteLine($"$Car is valid: {context.Validate()}");
    _showAlert = true;
  }
  public void StopOrder(EditContext context)
  {
    Console.WriteLine($"Car is invalid {string.Join(",",context.GetValidationMessages())}");
  }
}
```

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.

All files copyright Phil Japikse (http://www.skimedic.com/blog)