.NET 10 App Dev Hands-On Workshop

Blazor Lab 2 – Page/Component Life Cycle

This lab begins the work with ASP.NET Core Blazor WebAssembly (WASM). Before starting this lab, you must have completed Blazor Lab 1.

Copilot Agent Mode

```
Setup Prompt: Always use file scoped namespaces. Always combine attributes on a single line when
possible. The project does not use nullable reference types. If there is a GlobalUsings.cs file,
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. Don't add a constructor unless instructed to do so. Use primary
constructors when possible and don't declare a class level variable if the parameter from the
constructor can be used. Don't initialize properties unless instructed to do so. The @code block
in razor components should be at the bottom of the file. All work is to be done in the
AutoLot.Blazor project.

Prompt: Create a new folder named Services in the project, and in that folder create a public
class named OutputLog. Add a List<String> property named Messages to the OutputLog class.
Implement a method named LogMessage that accepts a string parameter and adds it to the Messages
list.

Prompt: Add a new file named GlobalUsings.cs in the root of the project, and add the following
global using statement to it (clear out any other code).
global using AutoLot.Blazor.Services;

Prompt: Add the OutputLog service to the dependency injection container in Program.cs with a
transient lifetime.

Prompt: Create new folder named Shared, and in that folder, create a Blazor component named
"LifecycleDemo.razor" in the that demonstrates the following lifecycle methods:
OnInitialized, OnParametersSet, SetParametersAsync, OnAfterRender, ShouldRender, OnAfterRender,
and Dispose. The component should accept a parameter named "Title" of type string and display it
in an <h3> tag. The component should accept a parameter named "DemoValue" of type int. The
component should also inject the OutputLog service and use it to log messages during each
lifecycle method. Each lifecycle method that has a callable base method, log a message to the
outputlog service indicating when it has been called just before calling the base method and just
after calling the base method, and the message should include the method name and "before" or
"after", accordingly. Lifecycle methods without a callable base method should just log a message
indicating when they have been called. Additionally, include a button that, when clicked, updates
a local property named "ClickCount" and displays the number of times the button has been clicked.
For the SetParametersAsync code, loop through the parameters and add them to the log. After the
button code, loop through the Messages in the OutputLog service and display them in an unordered
list. Also add a query string parameter named "initialCount" of type int with an initial value of
0. Ensure that the component properly cleans up any resources in the Dispose method.

Prompt: Add the following to the _Imports.razor file if they do not already exist:
@using AutoLot.Blazor.Shared
```

```
Prompt: Create a new razor page named Lifecycle.razor in the Pages folder that uses the
LifecycleDemo component. Make the route "/lifecycle-demo". The <PageTitle> value is "Lifecycle
Demo". Include a button that increments a local integer property named "DemoValue" and passes that
value to the LifecycleDemo component as a parameter.

Prompt: Add a menu item to the NavMenu.razor component that links to the new Lifecycle page with
the text "Lifecycle Demo".
Add a font-awsome icon "fa-solid fa-bolt-lightning" to the menu item. Add the icon before the
text. Use "pe-2" class on the icon for spacing.
```

Manual

Part 1: Add the Logger Utility

- Create a new folder named `services` in the `AutoLot.Blazor` project. In that folder, create a new class file named `OutputLog.cs`. Update the code to the following:

```
namespace AutoLot.Blazor.Services;

public class OutputLog
{
  public List<string> Messages { get; } = [];
  public void LogMessage(string message) => Messages.Add(message);
}
```

- Add a new file named `GlobalUsings.cs` to the root of the `AutoLot.Blazor` project. Update the new class to the following:

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

- Add the logger to the DI container by adding the following to the `Program.cs` file in `AutoLot.Blazor`:

```
builder.Services.AddTransient<OutputLog>();
```

Part 2: Add the Life Cycle Component

- Create a new folder named `Shared` in the `AutoLot.Blazor` project. In that folder, add a new Razor component named `LifeCycleDemo.razor`. Update the markup and the `@code` block to the following:

```
<h3>@Title</h3>
<button @onclick="OnClick">Click Me</button>
<span>Click count: @ClickCount</span>
<ul>
  @foreach (var message in OutputLog.Messages)
  {
    <li>@message</li>
  }
</ul>

@code {
  [Parameter] public string Title { get; set; }
  [Parameter] public int DemoValue { get; set; }
  [Inject] public OutputLog OutputLog { get; set; }
  [SupplyParameterFromQuery] public int InitialCount { get; set; } = 0;
  internal int ClickCount { get; set; }
  protected override void OnInitialized()
  {
    OutputLog.LogMessage("OnInitialized before base");
    base.OnInitialized();
    OutputLog.LogMessage("OnInitialized after base");
  }
  protected override void OnParametersSet()
  {
    OutputLog.LogMessage("OnParametersSet before base");
    base.OnParametersSet();
    OutputLog.LogMessage("OnParametersSet after base");
  }
  public override async Task SetParametersAsync(ParameterView parameters)
  {
    OutputLog.LogMessage("SetParametersAsync before base");
    foreach (var kvp in parameters)
    {
      OutputLog.LogMessage($"SetParametersAsync parameter: {kvp.Name} = {kvp.Value}");
    }
    await base.SetParametersAsync(parameters);
    OutputLog.LogMessage("SetParametersAsync after base");
  }
  protected override bool ShouldRender()
  {
    OutputLog.LogMessage("ShouldRender called before base");
    var result = base.ShouldRender();
    OutputLog.LogMessage($"ShouldRender after: {result}");
    return result;
  }
  protected override void OnAfterRender(bool firstRender)
  {
    OutputLog.LogMessage($"OnAfterRender before base: firstRender={firstRender}");
    base.OnAfterRender(firstRender);
    OutputLog.LogMessage($"OnAfterRender after base: firstRender={firstRender}");
  }
  public void Dispose()
  {
    OutputLog.LogMessage("Dispose called");
    // Clean up resources if needed
  }
  void OnClick()
  {
    ClickCount++;
  }
}
```

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

```
@using AutoLot.Blazor.Shared
```

Part 3: Add the Blazor Life Cycle Page

- In the `Pages` folder in `AutoLot.Blazor`, add a new Razor component named `LifeCycle.razor`. Update the markup and code to the following:

```
@page "/lifecycle-demo"
<PageTitle>Lifecycle Demo</PageTitle>
<h2>Lifecycle Demo</h2>
<button @onclick="IncrementDemoValue">Increment DemoValue</button>
<span>DemoValue: @DemoValue</span>
<LifecycleDemo Title="Lifecycle Demo Component" DemoValue="@DemoValue" />
@code {
  internal int DemoValue { get; set; }
  void IncrementDemoValue() { DemoValue++; }
}
```

- In the `Layout` folder in `AutoLot.Blazor`, update the `NavMenu.razor` component to add the new page to the menu. Add the following to the component after the `Home` menu item:

```
<div class="nav-item px-3">
  <NavLink class="nav-link" href="/lifecycle-demo" Match="NavLinkMatch.All">
    <span class="fa-solid fa-bolt-lightning pe-2" aria-hidden="true"></span> LifeCycle Demo
  </NavLink>
</div>
```

- Run the app to see the component/page life cycle events.

Summary

This completes the `BlazorLifeCycle` page.

Next Steps

The following lab will add the models and view models.