

# .NET 9 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.

### 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> Data = new List<string>();
    public void Log(string message)
    {
        Data.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 into 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 LifeCycle.razor. Update the @code block to the following:

```
@code {
    private bool _rendered;
    private bool _shouldRender;
    DateTime _created;
    private int _counter;
    [Parameter]
    public int Counter
    {
        get => _counter;
        set
        {
            _counter = value;
            CustomLogger.Log("-----");
            CustomLogger.Log($"Counter setter: set to {_counter}");
        }
    }
    public override Task SetParametersAsync(ParameterView parameters)
    {
        CustomLogger.Log("-----");
        CustomLogger.Log("SetParametersSetAsync called");
        _shouldRender = true;
        if (parameters.TryGetValue(nameof(Counter), out int counter))
        {
            // ignore odd values
            if (counter % 2 == 0)
            {
                _shouldRender = false;
            }
        }
        return base.SetParametersAsync(parameters);
    }
    protected override void OnInitialized()
    {
        _rendered = false;
        CustomLogger.Log("-----");
        CustomLogger.Log("OnInitialized In");
        base.OnInitialized();
        CustomLogger.Log("OnInitialized Out");
    }
    protected override void OnParametersSet()
    {
        CustomLogger.Log("-----");
        CustomLogger.Log("OnParametersSet In");
        base.OnParametersSet();
        CustomLogger.Log("OnParametersSet Out");
    }
    public void Dispose() => CustomLogger.Log("Disposed");
}
```

```

protected override bool ShouldRender()
{
    CustomLogger.Log("-----");
    CustomLogger.Log($"ShouldRender In");
    var result = base.ShouldRender();
    CustomLogger.Log($"ShouldRender: {result}");
    CustomLogger.Log($"ShouldRender Out");
    CustomLogger.Log($"***** PAGE END *****");
    return result;
}
protected override void OnAfterRender(bool firstRender)
{
    CustomLogger.Log("-----");
    CustomLogger.Log($"OnAfterRender In First:{firstRender}");
    base.OnAfterRender(firstRender);
    CustomLogger.Log($"OnAfterRender Out First:{firstRender}");
    if (!_rendered)
    {
        _rendered = true;
        StateHasChanged();
    }
}
//Optionally add in the async versions
//protected override async Task OnInitializedAsync()
//{
//    CustomLogger.Log("-----");
//    CustomLogger.Log("OnInitializedAsync in called");
//    await base.OnInitializedAsync();
//    CustomLogger.Log("OnInitializedAsync out called");
//}
//protected override async Task OnParametersSetAsync()
//{
//    CustomLogger.Log("-----");
//    CustomLogger.Log("OnParametersSetAsync in called");
//    await base.OnParametersSetAsync();
//    CustomLogger.Log("OnParametersSetAsync out called");
//}
//protected override async Task OnAfterRenderAsync(bool firstRender)
//{
//    CustomLogger.Log("-----");
//    CustomLogger.Log($"OnAfterRenderAsync in FirstRender = {firstRender}");
//    await base.OnAfterRenderAsync(firstRender);
//    CustomLogger.Log($"OnAfterRenderAsync out FirstRender = {firstRender}");
//}
}

```

- Update the markup to the following:

```

@inject OutputLog CustomLogger;
<h3>LifeCycle Events</h3>
@foreach (var line in CustomLogger.Data)
{
    @line
    <br />
}

```

- 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 `BlazorLifeCycle.razor`. Update the markup and code to the following:

```
@page "/blazor-life-cycle"
```

```
<PageTitle>LifeCycle</PageTitle>
```

```
<h3>Blazor LifeCycle</h3>
```

```
<LifeCycle Counter="@_counter"/>
```

```
<button @onclick="IncreaseCounter">Increase Counter</button>
```

```
@code
```

```
{
    int _counter = 1;

    public void IncreaseCounter()
    {
        _counter++;
    }
}
```

- 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="blazor-life-cycle" Match="NavLinkMatch.All">
        <span class="fa-solid fa-bolt-lightning pe-2" aria-hidden="true"></span> LifeCycle
    </NavLink>
</div>
```

- Update the `NavMenu.razor` component to use the Font Awesome font for the Home link (changes in bold):

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="blazor-life-cycle" Match="NavLinkMatch.All">
        <span class="fa-solid fa-home pe-2" aria-hidden="true"></span> Home
    </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.