Designing Reusable Components Using Advanced Blazor Features

Objective: By the end of this activity, you will implement advanced Blazor component features, including Dependency Injection, Cascading Parameters, and Component References, to create reusable and flexible components.

Step 1: Prepare for the Application

You'll create a new Blazor WebAssembly application and set up the necessary files and folders.

Instructions:

1. Open Visual Studio Code and ensure you have the .NET SDK (version 6.0 or later) installed on your system.
2. Open the terminal in Visual Studio Code by navigating to View > Terminal or pressing Ctrl + ~.
3. In the terminal, use the following command to create a new Blazor WebAssembly project: dotnet new blazorwasm -o AdvancedBlazorComponents
4. Navigate into the project folder by running: cd AdvancedBlazorComponents
5. Open the project in Visual Studio Code: code . This will open a new Visual Studio Code window. You will need to open the Terminal in Visual Studio Code for the next step.
6. Build and run the project to ensure it is working correctly by using: dotnet run
7. You should see Now listening on: http://localhost:5000 (your port number may be different)
8. Navigate to the Pages folder and create a new Razor file named Index.razor.
9. Add a routing directive to Index.razor to make it loadable.
10. Ensure that all default content is removed and add two instances of a component called ReusableComponent

Step 2: Implementing Dependency Injection

You will implement Dependency Injection to fetch data from a service and use it in a reusable component.

Instructions:

1. Create a Services folder in the project.
2. Add a service file called DataService.cs to fetch mock data.
3. In DataService.cs, create a function called GetData to return a list of strings.
4. Register the service in Program.cs.
5. Create a new Razor component named ReusableComponent.razor in the Pages folder.
6. Inject the service and use its data in ReusableComponent.razor.
7. Test the reusable component by using dotnet run and browse to http://localhost:5000/index to see the reusable component in index.razor.

Step 3: Utilizing Cascading Parameters

You will enhance the ReusableComponent by utilizing Cascading Parameters to share data.

Instructions:

1. Open the MainLayout.razor file in the Layout folder.
2. Wrap the layout content in a CascadingValue to define a cascading parameter called ThemeColor.
3. Open ReusableComponent.razor and use the cascading parameter to receive the ThemeColor value.
4. Create a <p> element with the style color set to the ThemeColor value.
5. Stop and restart your application and test index.razor with the updated reusable component.

Step 4: Leveraging Component References

You will add a child component and establish communication between parent and child using component references.

Instructions:

1. Create a new Razor file named ChildComponent.razor in the Pages folder.
2. Include the ChildComponent in ReusableComponent.razor and use a @ref directive to interact with it.

3. Test index.razor once more with the changes to reusable component.

Index.razor:

```
@page "/"
@using AdvancedBlazorComponents.Components

<h1 class="mb-4">Welcome to the Lab </h1>
<p>This page shows <strong>Dependency Injection</strong>,
<strong>Cascading Parameters</strong>, and <strong>Parent→Child</strong>
communication.</p>

<ReusableComponent />
```

ChildComponent.razor:

```
@using Microsoft.AspNetCore.Components
@namespace AdvancedBlazorComponents.Components

<div class="p-3 border rounded" style="border-color:@(AccentColor ??
"#d0d7de"); max-width:420px; margin:auto;">
    <h5 style="color:@(AccentColor ?? "inherit")">🧒 Child Counter</h5>
    <p>Count: <strong>@Count</strong></p>

    <div class="d-flex gap-2">
        <button class="btn btn-success btn-animate flex-fill"
@onclick="() => IncrementBy(1)">+1</button>
        <button class="btn btn-danger  btn-animate flex-fill"
@onclick="Reset">Reset</button>
    </div>
</div>

@code {
    [Parameter] public string? AccentColor { get; set; }
    public int Count { get; private set; }

    public void IncrementBy(int n)
    {
        Count += n;
        _ = InvokeAsync(StateHasChanged);
    }

    public void Reset()
    {
        Count = 0;
        _ = InvokeAsync(StateHasChanged);
    }
}
```

ReusableComponent.razor:

```
@using AdvancedBlazorComponents.Services
@inject IDataService DataService
@using AdvancedBlazorComponents.Components
```

```
<div class="card p-3 my-3 shadow-sm border rounded" style="max-
width:900px;">
    <h3>🎨 Tag Cloud (DI + Theme)</h3>

    <p style="color:@(ThemeColor ?? "inherit")">
        Theme color preview (current: @(ThemeColor ?? "default"))
    </p>

    @if (items is null)
    {
        <p>Loading...</p>
    }
    else
    {
        <div class="tag-cloud mb-3">
            @foreach (var tag in items)
            {
                var h = HueFrom(tag);
                <span class="badge me-1" style="background-
color:hsl(@(h), 70%, 80%)">@tag</span>
            }
        </div>
    }

    <button class="btn btn-outline-secondary mb-4" @onclick="Reload">🔄
Reload Tags</button>

    <hr />
    <h4>Parent → Child (via @@ref)</h4>
    <ChildComponent @ref="child" AccentColor="@(ThemeColor)" />

    <div class="d-flex gap-2 mt-2" style="max-width:420px;
margin:auto;">
        <button class="btn btn-primary flex-fill" @onclick="() =>
child?.IncrementBy(5)">Child +5</button>
        <button class="btn btn-warning flex-fill" @onclick="() =>
child?.Reset()">Reset Child</button>
        <button class="btn btn-outline-dark flex-fill"
@onclick="ReadChildCount">Read Count</button>
    </div>

    @if (lastRead.HasValue)
    {
        <p class="text-center mt-2">Parent read: Child.Count =
<strong>@lastRead</strong></p>
    }
</div>

@code {
    [CascadingParameter(Name = "ThemeColor")] public string? ThemeColor
{ get; set; }
    private List<string>? items;
    private ChildComponent? child;
    private int? lastRead;

    protected override async Task OnInitializedAsync()
        => items = await DataService.GetData();

    private async Task Reload() =>
```

```
        items = await DataService.GetData();

    private void ReadChildCount() =>
        lastRead = child?.Count;

    private int HueFrom(string s)
    {
        unchecked
        {
            int hash = 0;
            foreach (var c in s) hash = (hash * 31) + c;
            return Math.Abs(hash % 360);
        }
    }
}
```

IDataService.cs:

```
namespace AdvancedBlazorComponents.Services;

public interface IDataService
{
    Task<List<string>> GetData();
}
```

DataService.cs:

```
namespace AdvancedBlazorComponents.Services;

public class DataService : IDataService
{
    private readonly Random _rng = new();

    public Task<List<string>> GetData()
    {
        var techs = new List<string>
        {
            "Blazor", "C#", "Dependency Injection", ".NET 9",
"WebAssembly",
            "Reusable UI", "Razor Components", "CSS Animations",
            "Parent-Child Communication", "Bootstrap 5"
        };

        return Task.FromResult(techs.OrderBy(_ =>
_rng.Next()).ToList());
    }
}
```