Implementing Hybrid Rendering and Analyzing Blazor Application Performance

Activity: Implementing and Optimizing Blazor Application Performance

Objective: By the end of this lab, you will be able to implement hybrid rendering techniques in Blazor, analyze application performance using profiling tools, and apply optimization strategies to enhance application efficiency.

Step 1: Prepare for the Application

Create a new Blazor WebAssembly application using Visual Studio Code. This application will use hybrid rendering techniques and serve as the foundation for applying performance optimization techniques.

Instructions:

1. **Create a New Project:**
    a. Open a terminal in Visual Studio Code (Ctrl + `` or View > Terminal).
    b. Run the following command to create a new Blazor WebAssembly project: dotnet new blazorwasm -o BlazorPerformanceApp This will create a new folder named BlazorPerformanceApp with the required files.
2. **Open the Project:**
    a. Navigate into the project directory: cd BlazorPerformanceApp
    b. Open the project in Visual Studio Code: code .
3. **Run the Application:**
    a. In the terminal, run the application: dotnet run
    b. Copy the URL displayed in the terminal (e.g., http://localhost:5000) and open it in your browser.
    c. Verify that the default Blazor WebAssembly application loads successfully.
4. **Clean Up the Default Code:**
    a. Open the Pages folder and review the default Razor components.
    b. Identify the file named Home.razor in the Pages folder. This is the file you will modify in subsequent steps.

Step 2: Implement Hybrid Rendering

Add hybrid rendering logic by combining server-side and client-side Blazor features in the HybridComponent.

Instructions:

1. In the Pages folder, create a new Razor component named HybridComponent.razor.
2. Use mock data to simulate a hybrid rendering scenario.
3. Add the HybridComponent to Home.razor to verify it renders correctly.
4. Now we'll need to update the Program.cs file:
    a. After the variable builder is declared and before the line await builder.Build().RunAsync(); add this line: builder.Services.AddInteractiveServerComponents();

Step 3: Monitor and Analyze Performance

Use .NET's built-in diagnostics tools to measure the performance of your Blazor application.

Instructions:

- Add Logging: Inject logging functionality into the HybridComponent.
- Log Key Events: Modify the OnInitializedAsync method in HybridComponent.razor to log key performance events

Step 4: Apply optimization Techniques

Optimize the HybridComponent for performance by adding lazy loading functionality.

Instructions:

- Modify Home.razor to Include Lazy Loading: Update Home.razor by replacing the direct inclusion of the HybridComponent with a lazy loading approach.
- Test the Optimization:
    - Rebuild and run the application.
    - Click the "Load Hybrid Component" button to verify the lazy loading functionality.

Step 5: Validate and Reassess

Re-evaluate the application after optimizations to ensure performance has improved.

Instructions:

1. Rerun Performance Measurements: Observe the application's behavior before and after clicking the "Load Hybrid Component" button.
2. Document Observations:
    a. Note any improvements in responsiveness and behavior.
    b. Discuss how lazy loading improves performance by deferring resource usage until required.

Home.razor:

```
@page "/"
@rendermode InteractiveServer
@using static Microsoft.AspNetCore.Components.Web.RenderMode
@using BlazorPerf.Components
@using BlazorPerf.Client.Components
@inject NavigationManager Nav

<PageTitle>Home</PageTitle>

<section class="hero">
  <div class="hero__content">
    <h1 class="title">Blazor Hybrid Rendering Lab</h1>
    <p class="subtitle">SSR • Interactive Server • Interactive
WebAssembly</p>
    <button class="btn" @onclick="LoadHybrid">Load Hybrid
Inline</button>
    <a class="btn btn--ghost" href="/hybrid" style="margin-
left:8px;">Open Full Page</a>
  </div>
</section>

<section class="preview">
  <h2 class="preview__title">Live Preview</h2>
  <div class="cards">
    <div class="card">
      <h3>SSR-only</h3>
      <SSRPane />
    </div>
    <div class="card">
      <h3>Interactive Server</h3>
      <ServerClock @rendermode="InteractiveServer" />
    </div>
  </div>
</section>

@if (showHybrid)
{
```

```
  <section class="lazy" id="hybrid-host">
    <h2 class="preview__title">Hybrid (lazy)</h2>
    <article class="lazy__host">
      <HybridComponent />
    </article>
  </section>
}

@code {
    private bool showHybrid;
    private async Task LoadHybrid()
    {
        if (!showHybrid) { showHybrid = true; await Task.Yield(); }
        Nav.NavigateTo("#hybrid-host");
    }
}
```

HybridComponent.razor:

```
@using System.Diagnostics
@page "/hybrid"
@rendermode InteractiveServer
@using static Microsoft.AspNetCore.Components.Web.RenderMode
@using BlazorPerf.Components
@using BlazorPerf.Client.Components
@inject ILogger<HybridComponent> Logger

<PageTitle>Hybrid Component</PageTitle>

<section class="hybrid">
  <header class="hybrid__head">
    <h1>Hybrid Component</h1>
  </header>

  <div class="cards">
    <div class="card">
      <h3>SSR-only</h3>
      <SSRPane />
    </div>
    <div class="card">
      <h3>Interactive Server</h3>
      <ServerClock @rendermode="InteractiveServer" />
    </div>
    <div class="card">
      <h3>Interactive Auto (WASM)</h3>
      <SlowList @rendermode="InteractiveAuto" ItemsCount="300" />
    </div>
  </div>
</section>

@code {
    private readonly Stopwatch _sw = new();

    protected override void OnInitialized()
    {
        _sw.Start();
        Logger.LogInformation("HybridComponent OnInitialized at {Utc}",
DateTime.UtcNow);
```

```
    }

    protected override async Task OnInitializedAsync()
    {
        Logger.LogInformation("HybridComponent OnInitializedAsync start
at {Utc}", DateTime.UtcNow);
        await Task.Yield();
        Logger.LogInformation("HybridComponent OnInitializedAsync end at
{Utc} (elapsed {Ms} ms)", DateTime.UtcNow, _sw.ElapsedMilliseconds);
    }

    protected override Task OnAfterRenderAsync(bool firstRender)
    {
        if (firstRender)
            Logger.LogInformation("HybridComponent first render complete
at {Utc} (elapsed {Ms} ms)", DateTime.UtcNow, _sw.ElapsedMilliseconds);
        return Task.CompletedTask;
    }
}
```

ServerClock.razor:

```
@using System.Timers
@implements IDisposable

<div class="element">
    <p>Server time</p>
    <h3>@_now.ToLongTimeString()</h3>
</div>

@code {
    private DateTime _now = DateTime.Now;
    private Timer? _timer;

    protected override void OnInitialized()
    {
        _timer = new Timer(1000);
        _timer.Elapsed += (_, __) =>
        {
            _now = DateTime.Now;
            InvokeAsync(StateHasChanged);
        };
        _timer.Start();
    }

    public void Dispose()
    {
        _timer?.Stop();
        _timer?.Dispose();
    }
}
```

SSRPane.razor:

```
@code {
    private readonly DateTime _renderedAt = DateTime.UtcNow;
```

```
    private readonly string[] _mock = new[] { "Alpha", "Bravo",
"Charlie", "Delta" };
}
<div class="element">
    <p>Rendered at (UTC): @_renderedAt:HH:mm:ss</p>
    <ul>
        @foreach (var s in _mock)
        {
            <li>@s</li>
        }
    </ul>
</div>
```

SlowList.razor:

```
@code {
    [Parameter] public int ItemsCount { get; set; } = 50;
    private List<string> _items = new();

    protected override void OnInitialized()
    {
        for (int i = 0; i < ItemsCount; i++)
        {
            _items.Add($"Item {i:D3}");
        }
    }

    private void Shuffle()
    {
        var rnd = new Random();
        for (int i = _items.Count - 1; i > 0; i--)
        {
            int j = rnd.Next(i + 1);
            (_items[i], _items[j]) = (_items[j], _items[i]);
        }
    }
}
<div class="element">
    <button class="btn" @onclick="Shuffle">Shuffle</button>
    <ul>
        @foreach (var it in _items)
        {
            <li>@it</li>
        }
    </ul>
</div>
```