Implementing Server-Side Blazor Applications

Activity: create a Server-Side Blazor Application with SignalR and State Management

Objective: by the end of this lab, you will create a server-side Blazor application using Visual Studio Code. This activity demonstrates real-time communication using SignalR and state management features in a Blazor Server application. You will also modify the default navigation to include links to your new pages.

Step 1: Prepare for the Application

Set up a server-side Blazor application using the updated blazor template.

Instructions:

1. Open Visual Studio Code and launch the terminal.
2. Use the Blazor template to create a server-side Blazor application: dotnet new blazor -o BlazorServerApp
3. Change to the newly created application directory: cd BlazorServerApp
4. Open the folder in Visual Studio Code: code .
5. Restore dependencies: dotnet restore
6. Install the SignalR client package: dotnet add package Microsoft.AspNetCore.SignalR.Client --version 8.*
7. Run the application: dotnet run
8. Open your browser at the location indicated in the terminal to confirm the app runs correctly.

Step 2: Configure Real-Time Features with SignalR

Implement a real-time chat feature using SignalR.

Server-Side Setup

1. In the Program.cs file, configure the SignalR services: builder.Services.AddSignalR(); app.MapHub<NotificationHub>("/notificationHub");
2. Create a folder named Hubs in the root directory.

3. Inside the Hubs folder, create a new file named NotificationHub.cs and add code to define a SignalR hub that broadcasts messages from a user to all connected clients in real time.

Client-Side Integration

1. Navigate to the Components/Pages folder.
2. Create a new Razor component named SignalRChat.razor in the Components/Pages folder.
3. Add code to SignalRChat.razor that implements a real-time chat component in Blazor Server using SignalR to enable users to send and receive messages dynamically without page refresh.

Step 3: Demonstrate State Management

Add a page to demonstrate server-side state management.

Instructions:

1. Create a new Razor component named StateManagement.razor in the Components/Pages folder.
2. Add code to StateManagement.razor to increment a counter.

Step 4: Update Navigation

Modify the navigation menu to include links to the new pages.

Instructions:

1. Open the Components/Layout/NavMenu.razor file.
2. Add links to the SignalRChat and StateManagement pages.

Step 5: Test the Application

Instructions:

1. Run the application: dotnet run
2. Navigate to:
    - /signalrchat: Test the 0real-time chat functionality.

- /state: Test the state management functionality by clicking the button and observing the counter.

SignalRChat.razor:

```
@page "/signalr-chat"
@rendermode InteractiveServer
@using Microsoft.AspNetCore.SignalR.Client
@inject NavigationManager Nav

<h3>SignalR Chat</h3>

<div class="mb-3">
    <label class="form-label">Display name</label>
    <input @bind="userName" @bind:event="oninput" class="form-control"
/>
</div>

<div class="mb-3">
    <label class="form-label">Message</label>
    <input @bind="message" @bind:event="oninput" class="form-control"
@onkeydown="HandleEnter" />
    <button class="btn btn-primary mt-2"
@onclick="SendAsync">Send</button>
</div>

<p class="text-muted">Hub state: @hubState</p>

<ul class="list-group">
    @foreach (var m in messages)
    {
        <li class="list-group-item">
            <b>@m.User</b>: @m.Text
            <div class="text-muted" style="font-
size:.8rem">@m.When.LocalDateTime</div>
        </li>
    }
</ul>

@code {
    private HubConnection? hub;
    private string? userName;
    private string? message;
    private string hubState = "Not connected";

    private readonly List<ChatMessage> messages = new();

    protected override async Task OnInitializedAsync()
    {
        hub = new HubConnectionBuilder()
            .WithUrl(Nav.ToAbsoluteUri("/notificationHub"))
            .WithAutomaticReconnect()
            .Build();

        hub.Reconnecting += error => { hubState = "Reconnecting...";
StateHasChanged(); return Task.CompletedTask; };
        hub.Reconnected  += id    => { hubState = "Connected";
StateHasChanged(); return Task.CompletedTask; };
```

```
        hub.Closed       += error => { hubState = "Closed";
StateHasChanged(); return Task.CompletedTask; };

        hub.On<string, string, DateTimeOffset>("ReceiveMessage", (user,
text, when) =>
        {
            messages.Add(new ChatMessage(user, text, when));
            InvokeAsync(StateHasChanged);
        });

        await hub.StartAsync();
        hubState = "Connected";
    }

    private async Task SendAsync()
    {
        if (hub is null) return;

        var u = string.IsNullOrWhiteSpace(userName) ? "Anonymous" :
userName!.Trim();
        var t = message?.Trim();
        if (string.IsNullOrWhiteSpace(t)) return;

        await hub.SendAsync("SendMessage", u, t);
        message = string.Empty;
    }

    private async Task HandleEnter(KeyboardEventArgs e)
    {
        if (e.Key == "Enter")
        {
            await SendAsync();
        }
    }

    public async ValueTask DisposeAsync()
    {
        if (hub is not null)
        {
            await hub.DisposeAsync();
        }
    }

    private record ChatMessage(string User, string Text, DateTimeOffset
When);
}
```

StateManagement.razor:

```
  @page "/state-management"
@rendermode InteractiveServer
@inject BlazorServerApp.Services.StateContainer State

<h3>Server-Side State Management</h3>

<p class="lead">
    Counter value: <b>@State.Counter</b>
</p>
```

```
<button class="btn btn-success" @onclick="Increment">+1</button>

@code {
    protected override void OnInitialized()
    {
        State.OnChange += StateHasChanged;
    }

    private void Increment() => State.Increment();

    public void Dispose()
    {
        State.OnChange -= StateHasChanged;
    }
}
```

NotificationHub.cs:

```
using Microsoft.AspNetCore.SignalR;

namespace BlazorServerApp.Hubs;

public class NotificationHub : Hub
{
    public async Task SendMessage(string user, string message)
    {
        var safeUser = string.IsNullOrWhiteSpace(user) ? "Anonymous" :
user.Trim();
        var safeMsg  = message?.Trim() ?? string.Empty;

        await Clients.All.SendAsync(
            "ReceiveMessage",
            safeUser,
            safeMsg,
            DateTimeOffset.UtcNow
        );
    }
}
```

StateContainer.cs:

```
namespace BlazorServerApp.Services;

public class StateContainer
{
    public int Counter { get; private set; }
    public event Action? OnChange;

    public void Increment()
    {
        Counter++;
        NotifyStateChanged();
    }

    private void NotifyStateChanged() => OnChange?.Invoke();
}
```