Handling API Responses and State Management in Blazor

Step 1: Extend the Blazor Application

You'll extend the Blazor WebAssembly application from the previous lab to incorporate dynamic state management for API data updates.

Instructions:

1. Open the WeatherApp project you created in the previous activity.
2. Verify the WeatherFetch.razor component is functioning and correctly fetching data from the weather API.
3. Create a new Razor component named DynamicWeather.razor in the Pages folder to handle dynamic updates.

Step 2: Fetch and Deserialize a JSON Response

You will fetch and deserialize user data from a public API (JSONPlaceholder) and integrate it into your Blazor component.

Instructions:

1. Identify a secondary API to fetch data. For example:
    - Fetch placeholder user data from https://jsonplaceholder.typicode.com/users.
2. Create a Models Folder
3. Add the User Class to model the JSONPlaceholder API response.
4. Inject HttpClient.
5. Update the DynamicWeather.razor component to add a method to fetch user data dynamically, including CancellationToken support to cancel overlapping requests.
    - Use HttpClient to fetch and deserialize the JSON response from the API in the OnInitializedAsync method. Include a CancellationToken to cancel previous API calls when a new one is initiated.

Step 3: Implement State Management to Handle API Data Updates

Add a service to manage the shared state for API data and integrate it with the Blazor component.

Instructions:

1. Add a WeatherData class to the Models folder to model the weather API response.
2. Add a WeatherStateService class to manage the shared state in the Services folder (create one if it doesn't exist):
    - Use EventCallback or Action delegates to notify components of state changes.
    - Store the weather data and user data in the service.
3. Register the state service in Program.cs: builder.Services.AddSingleton<WeatherStateService>();
4. Inject the service into DynamicWeatherFetch.razor to use and update shared state.
5. Add the StateHasChanged subscription to ensure the UI updates when the state changes.

Step 4: Test UI Updates Dynamically Based on New API Responses

Ensure the UI updates dynamically when the state changes.

Instructions:

1. Add buttons or triggers to fetch new data from both APIs dynamically.
2. Use StateHasChanged() to trigger UI updates in the OnInitialized method to respond to dynamic updates, and invoke StateHasChanged() after updating the state service in OnInitializedAsync.
3. Display the weather and user data in the component.
4. Implement a loading indicator to show the user when an API call is in progress.