Implementing State Management and Form Handling/Validation

By the end of this activity, learners will create a Blazor application that demonstrates state management and includes a form with validation to collect user feedback.

Step 1: Prepare for the Application

You'll create a new Blazor WebAssembly project to manage user feedback. The application will allow users to submit feedback via a form, leveraging Blazor's state management and validation capabilities.

1. Open Visual Studio Code.
2. Create a new Blazor WebAssembly project:
    a. Navigate to the project directory: dotnet new blazorwasm -o FeedbackApp
    b. Open the terminal and run: cd FeedbackApp
3. Open the project in Visual Studio Code.

Step 2: Set up the Data Model

Define a data model to store feedback details, including validation attributes.

Instructions:

1. Create a folder named Models in the project root.
2. Inside the Models folder, create a file named Feedback.cs.
3. Define a class Feedback with the following properties:
    a. Name: A required string property for the user's name.
    b. Email: A string property with an email format validation.
    c. Comment: A string property with a maximum length validation.

Step 3: Build the Feedback Form

Create a form to capture user feedback.

Instructions:

1. Open the Pages folder and create a new Razor component file named FeedbackForm.razor.

2. Use Blazor components <EditForm>, <InputText>, <InputTextArea>, and <ValidationSummary> to create the form.
3. Bind the form inputs to the properties in the Feedback model.
4. Use <DataAnnotationsValidator> to enable validation.

Step 4: Implement State Management

Store and manage the submitted feedback in the application state.

Instructions:

1. Create a new folder named Services.
2. Add a file named FeedbackService.cs in the Services folder.
3. Define a service to manage a list of feedback entries:
    a. Provide methods to add and retrieve feedback.
4. Register the service in Program.cs using dependency injection.

Step 5: Handle Form Submission

Process and validate the submitted feedback.

Instructions:

1. Add a method to the FeedbackForm.razor component to handle the form submission event.
2. Use dependency injection to access the FeedbackService and add the submitted feedback.
3. Clear the form after successful submission and display a confirmation message.

Step 6: Display Submitted Feedback

Create a component to display the list of feedback.

Instructions:

1. Create a new Razor component named FeedbackList.razor.
2. Use a Blazor <table> to display feedback stored in the service.
3. Retrieve feedback data from the FeedbackService.

Models, FeedbackItem.cs:

```
using System.ComponentModel.DataAnnotations;

namespace FeedbackApp.Models
{
    public class FeedbackItem
    {
        [Required(ErrorMessage = "Name is required")]
        public string Name { get; set; } = string.Empty;

        [EmailAddress(ErrorMessage = "Invalid email format")]
        public string Email { get; set; } = string.Empty;

        [MaxLength(500, ErrorMessage = "Comment cannot exceed 500
characters")]
        public string Comment { get; set; } = string.Empty;
    }
}
```

FeedbackForm.razor:

```
@page "/feedback"
@using FeedbackApp.Models
@using FeedbackApp.Services
@inject IFeedbackService FeedbackService

<h3>Submit Feedback</h3>

<EditForm Model="@model" OnValidSubmit="HandleValidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <div class="mb-3">
        <label class="form-label">Name</label>
        <InputText @bind-Value="model.Name" class="form-control" />
        <ValidationMessage For="@(() => model.Name)" />
    </div>

    <div class="mb-3">
        <label class="form-label">Email</label>
        <InputText @bind-Value="model.Email" class="form-control" />
        <ValidationMessage For="@(() => model.Email)" />
    </div>

    <div class="mb-3">
        <label class="form-label">Comment</label>
        <InputTextArea @bind-Value="model.Comment" class="form-control"
rows="5" />
        <ValidationMessage For="@(() => model.Comment)" />
    </div>

    <button type="submit" class="btn btn-primary">Submit</button>
</EditForm>

@if (submitted)
{
    <div class="alert alert-success mt-3">
        Thanks for your feedback, @lastSubmittedName!
    </div>
}

@code {
    private FeedbackItem model = new();
    private bool submitted;
    private string lastSubmittedName = string.Empty;

    private void HandleValidSubmit()
    {
        FeedbackService.Add(model);
        lastSubmittedName = model.Name;
        submitted = true;
        model = new();
    }
}
```

FeedbackList.razor:

```
@page "/feedback/list"
@using FeedbackApp.Models
@using FeedbackApp.Services
@inject IFeedbackService FeedbackService

<h3>Feedback List</h3>

<div class="mb-3">
    <NavLink class="btn btn-primary" href="feedback">+ Add
Feedback</NavLink>
</div>

@if (items.Count == 0)
{
    <p>No feedback yet.</p>
    <NavLink class="btn btn-outline-primary" href="feedback">Create
first feedback</NavLink>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Name</th>
                <th>Email</th>
                <th>Comment</th>
            </tr>
        </thead>
        <tbody>
        @foreach (var f in items)
        {
            <tr>
                <td>@f.Name</td>
                <td>@f.Email</td>
                <td>@f.Comment</td>
            </tr>
        }
        </tbody>
    </table>
}

@code {
    private IReadOnlyList<FeedbackItem> items =
Array.Empty<FeedbackItem>();

    protected override void OnInitialized()
    {
        items = FeedbackService.GetAll();
    }
}
```

FeedbackService.cs

```
using System.Collections.ObjectModel;
using FeedbackApp.Models;

namespace FeedbackApp.Services;

public class FeedbackService : IFeedbackService
{
    private readonly ObservableCollection<FeedbackItem> _items = new();

    public void Add(FeedbackItem item)
    {
        if (item is null) return;
        _items.Add(new FeedbackItem
        {
            Name = item.Name,
            Email = item.Email,
            Comment = item.Comment
        });
    }

    public IReadOnlyList<FeedbackItem> GetAll()
        => new ReadOnlyCollection<FeedbackItem>(_items);
}
```

IFeedbackService.cs:

```
using FeedbackApp.Models;

namespace FeedbackApp.Services;

public interface IFeedbackService
{
    void Add(FeedbackItem item);
    IReadOnlyList<FeedbackItem> GetAll();
}
```

Program.cs:

```
builder.Services.AddScoped<IFeedbackService, FeedbackService>();
```