Add icon   Add cover

State-Record Pattern in Blazor Pages

This is like the evolution of the Is-Loading Pattern in Blazor Pages note, but on steroids. Instead of a binary state (`IsLoading` is true or false), we can have multiple states. No null warnings, no question marks or exclamation points are required.

Define possible states for a page

Here we define all possible states for a page, and each state can have multiple properties.

`BlazorPageExample.razor.cs`

C#

```csharp
using Microsoft.AspNetCore.Components;

namespace ExampleNamespace
{
    public class BlazorPageExampleBase : ComponentBase
    {
        protected abstract record State
        {
            // This state has no extra properties.
            public record Loading : State;

            // This state has some meaningful properties.
            // We can be confident they won't be null if the state is "FormEdit"
            // The "Model" property is only relevent in the "FormEdit" state,
            // so null checking is not needed.
            // The properties "Model" and "IsSubmitting" are implicitly defined using this syntax.
            public record FormEdit(ExampleDto Model, bool IsSubmitting = false) : State;

            // Empty parenthesis are optional
            public record Success() : State;

            // The record can of course have a more complicated definition, with explicit properties.
            // The curly brace body is optional.
            public record Failure(string ErrorMessage) : State
            {
                public readonly string ErrorMessage = ErrorMessage;

                // This function is only available in the Failure state.
                // It's impossible to use this function in the incorrect state.
                public void GoHome()
                {
                  //...
                }
            }
        }

        protected State CurrentState { get; private set; } = new State.Loading();

        protected override void OnInitialized()
        {
            var model = new ExampleDto("test");

            CurrentState = new State.FormEdit(model);
        }

        // This function is only available in the "FormEdit" state,
        // so State.FormEdit is required as a parameter.
        protected async Task OnSubmit(State.FormEdit formState)
        {
            try
            {
                await Task.CompletedTask; // Call some api service
                CurrentState = new State.Success();
            }
            catch (Exception ex)
            {
                CurrentState = new
```

```
State.Failure(ex.Message);
            }
        }
    }

    public class ExampleDto(string name)
    {
        public string Name { get; set; } = name;
    }
}
```

And now in the razor page, we can use pattern matching with if statements to both check the state and cast it. You can also use a `switch` expression instead of `if` statements.

`BlazorPageExample.razor`

```
@inherits BlazorPageExampleBase

<h1>Example Header</h1>

@if (CurrentState is State.Loading)
{
    <LoadingSpinner />
}
else if (CurrentState is State.FormEdit formEditState)
{
    <EditForm Model="formEditState.Model" OnValidSubmit="() => OnSubmit(formEditState)">
        @* Name *@
        <div class="col-12">
            <div class="form-floating mb-3">
                <input id="Name" type="text" @bind="formEditState.Model.Name" class="form-control" />
                <label for="Name">Name</label>
                <ValidationMessage For="@(() => formEditState.Model.Name)" />
            </div>
        </div>
    </EditForm>
}
else if (CurrentState is State.Success)
{
    <h2>Yay!</h2>
}
```

```
else if (CurrentState is State.Failure failState)
{
    <h2>Oh no!</h2>
    <span>@failState.ErrorMessage</span>
}
```

I got this idea from learning a bit about rust, and finding this reddit post:

https://www.reddit.com/r/csharp/comments...