Working with Blazor Components

Gill Cleeren
CTO Xpirit Belgium

@gillcleeren – xpirit.com/gill

Module overview

- Doing more with components
- Adding navigation
- Using RenderFragment
- Loading components dynamically
- Handling errors in components
- Using built-in components

Doing More with Components

Displaying Data in a Component UI

Technically data binding: one-way data binding from source to target (UI)

EmployeeDetail.razor

```
<h1 class="page-title">
    Details for @FirstName @LastName
</h1>
```

EmployeeDetail.razor.cs

```
public string FirstName { get; set; }
public string LastName { get; set; }
```

Changing the Access Modifier

EmployeeDetail.razor

```
<h1 class="page-title">
    Details for @_firstName @_lastName
</h1>
```

EmployeeDetail.razor.cs

```
private string _firstName;
private string _lastName;
```

Dotting into the Properties

EmployeeDetail.razor

```
<h1 class="page-title">
    Details for
    @Employee.FirstName
    @Employee.LastName
</h1>
```

EmployeeDetail.razor.cs

```
public Employee Employee { get; set; }
```

Nesting Components

Components can include other components

Declared in HTML

Nesting Components

May require @using to be added

PageHeader.razor

```
<h1>@PageTitle</h1>
```

EmployeeOverview.razor

```
@page "/employeeoverview"

<PageHeader></PageHeader>
```

Components live in a namespace

Root namespace + folder

- Typically project name

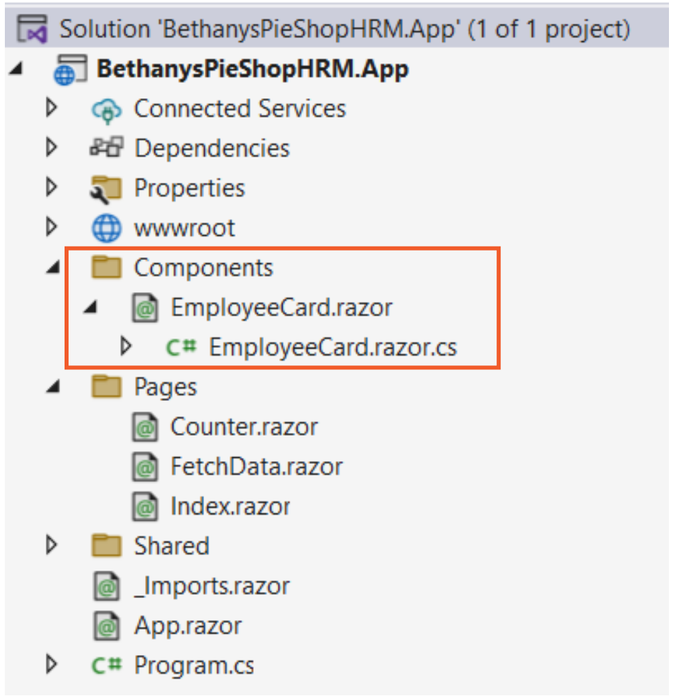
May require @using to be added when using the component

```
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web

@using BethanysPieShopHRM.App.Components
```

Using the _Imports.razor File

Component Namespaces

```
@using BethanysPieShopHRM.App.Components
```



Component Parameters

Parameters are used to pass data between components

Use the [Parameter] attribute

Parameter can be simple or complex type

Accepting a Parameter

```
<h3>@Name</h3>



@code {
    [Parameter]
    public string Name { get; set; } = string.Empty;
}
```

```
<EmployeeCard Name="Gill Cleeren"></EmployeeCard>
```

Invoking a Component with a Parameter

Demo

Creating the employee card component

Passing data using [Parameter]

Working with Events in Components

```
@on{Dom Event}="Delegate"
```

```
<button @onclick="SaveEmployee">Save</button>

@code {
   private void SaveEmployee()
   {
      //save the employee to the backend

   }
}
```

```
<button @onclick="ShowLocation">Show</button>

@code {
    private void ShowLocation(MouseEventArgs e)
    {

    }
}
```

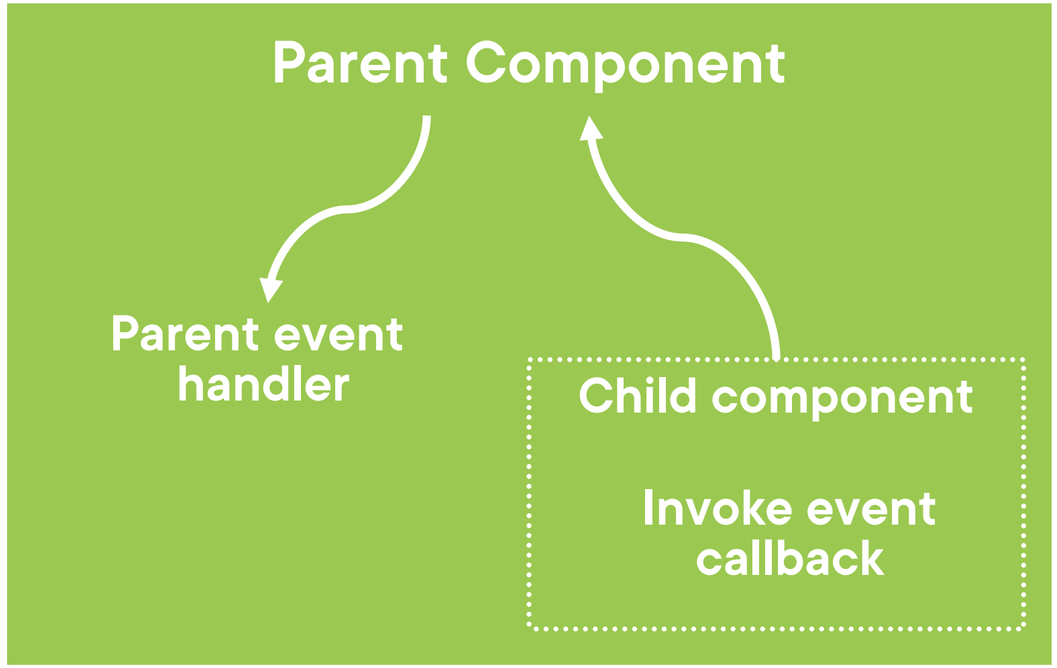
Using the Default Event Arguments

@onclick passes MouseEventArgs
@onkeydown passes KeyboardEventArgs

Using Lambda Expressions

```
@for (int i = 1; i < 10; i++)
{
    var buttonNumber = i;

    <p>
        <button @onclick="@(e => ShowLocation(e, buttonNumber))">
            Button @i
        </button>
    </p>
}

@code {

    private void ShowLocation(MouseEventArgs e, int buttonNumber)
    {

    }
}
```

Using EventCallback



```
<button @onclick="TriggerCallbackToParent">Show</button>

@code {

    [Parameter]
    public EventCallback<MouseEventArgs> TriggerCallbackToParent { get; set; }
}
```

Using EventCallback in the Nested Child Component

```
<ChildComponent TriggerCallbackToParent="ShowPopup"></ChildComponent>

@code {
    private void ShowPopup()
    {
        ...
    }
}
```

Reacting to an EventCallback in the Parent Component

Working with the Component Lifecycle

Events which are triggered automatically at certain points

Write code in overrides to hook into these

Important Lifecycle Events

OnInitialized()
OnInitializedAsync()

OnParametersSet()
OnParametersSetAsync()

OnAfterRender()
OnAfterRenderAsync()

```
protected override void OnInitialized()
{
    //Initialization code for the component
}
```

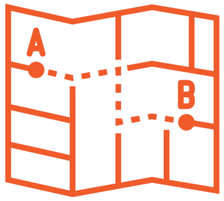
Overriding a Lifecycle Event

Demo

Creating the Quick View popup component

Adding Navigation

Navigating in Blazor Applications



Router in App.razor is starting point

@page directive enables routing to the component

[1,2,3]

Can accept parameters

Use NavigationManager for code-based navigation

Router in App.razor

```
<Router AppAssembly="@typeof(App).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
    </Found>
    <NotFound>
       <h1>We can't find your page...</h1>
    </NotFound>
</Router>
```

```
@page "/employeeoverview"
```

A Component We Can Navigate to

```
@page "/employeeoverview“
@page "/employeelist"
```

Multiple Page Attributes

```
@page "/employeedetail/{EmployeeId}“


[Parameter]
public string EmployeeId { get; set; }
```

Adding Route Parameters

```
@page "/employeedetail/{Id:int}“


[Parameter]
public int Id { get; set; }
```

Adding a Constraint

```
[Parameter]
[SupplyParameterFromQuery(Name = "id")]
public string EmployeeId { get; set; }
```

SupplyParameterFromQuery

Specify that value can come from query string
Name property can be used to define a different query parameter

```
[Inject]
public NavigationManager NavigationManager { get; set; }


NavigationManager.NavigateTo($"/employeedetail/{selectedEmployee.EmployeeId}");
```

Triggering Navigation from Code

NavigationManager is injected here using dependency injection

Demo

Adding the employee details component

Navigating to the details page

Using RenderFragment

Setting the Content

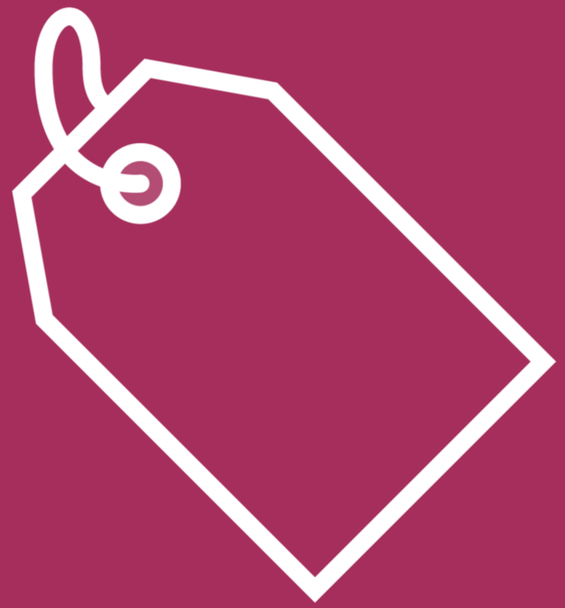
<ProfilePicture>actual-image-name</ProfilePicture>

ProfilePicture

actual-image-name

Using RenderFragment

```
<div class="profile-picture">
    @ChildContent
</div>

@code {

    [Parameter]
    public RenderFragment? ChildContent { get; set; }
}
```



Naming is everything!

The RenderFragment property must be named ChildContent!

Demo

Using RenderFragment to pass content

Loading Components Dynamically

Using Dynamic Components

```
<DynamicComponent Type="@type" />

@code {

    private Type type = ...;
}
```

Using DynamicComponent

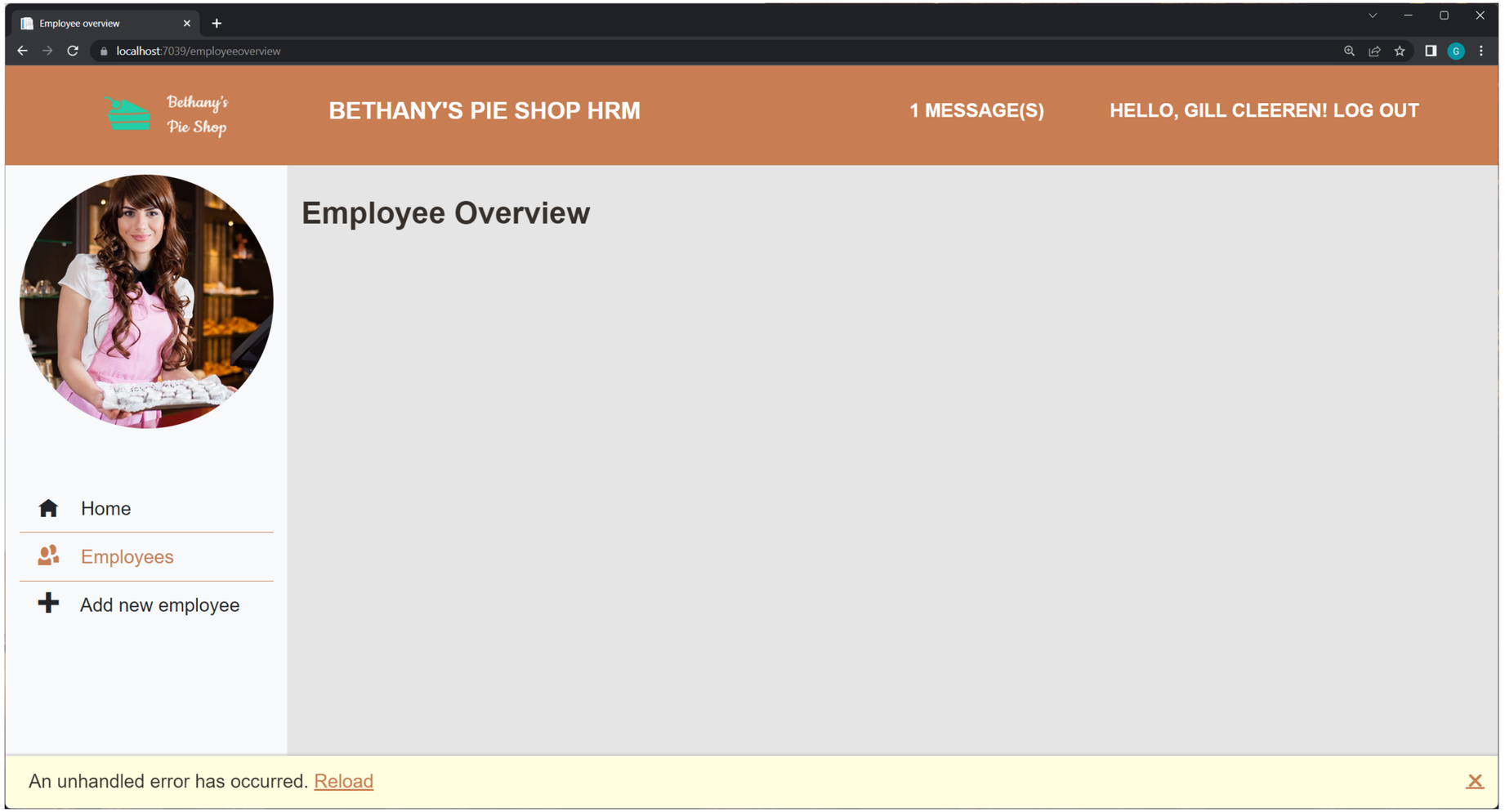
Possible to pass in parameters too
Can render a dynamic UI if used in a loop

Demo

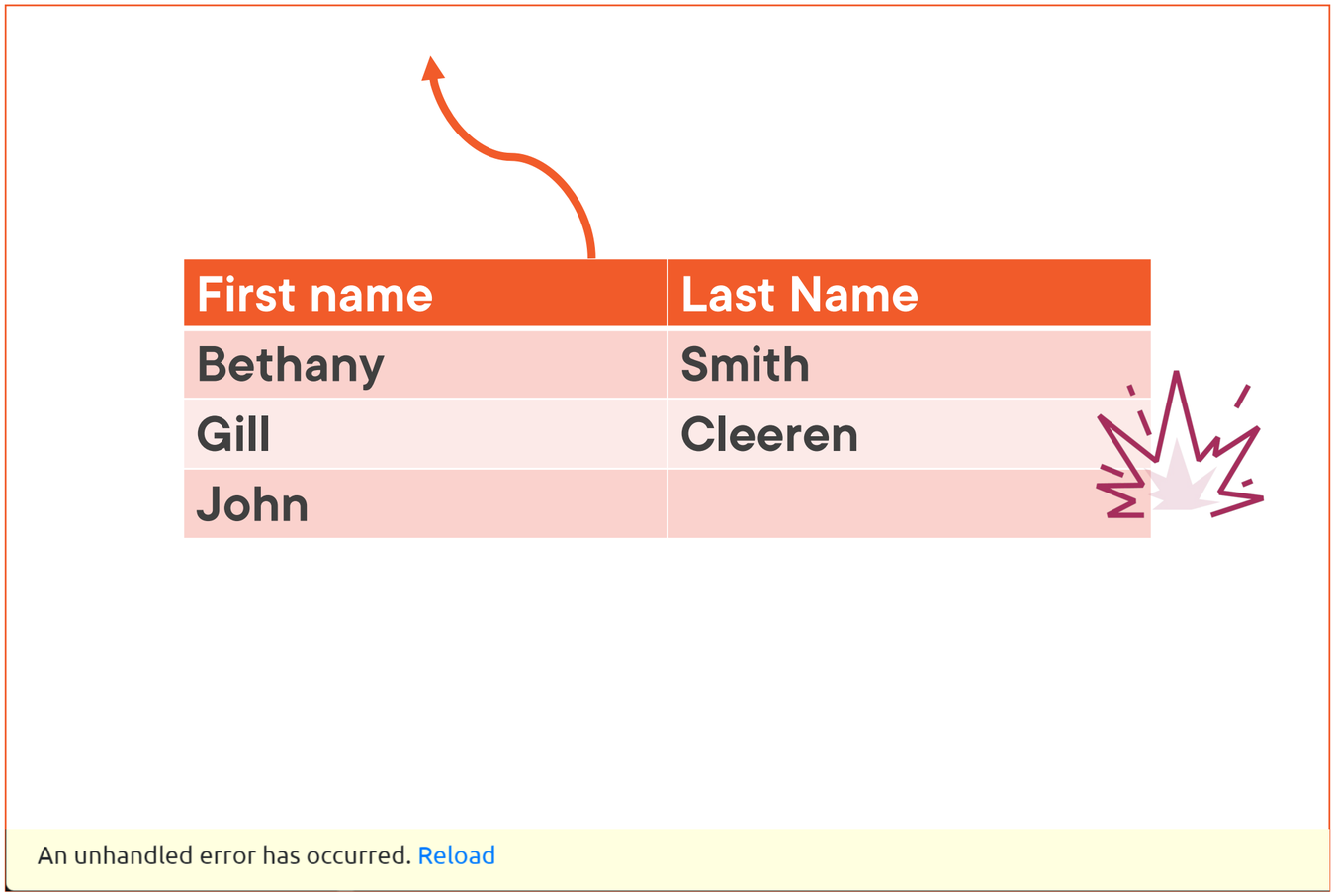
Loading widgets on the home page

Handling Errors in Components

The Default Exception Handling in Components



Default Exception Handling



Adding Error Boundaries

First name	Last Name
Bethany	Smith
Gill	Cleeren
John	

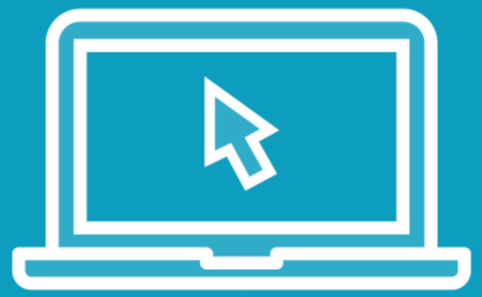
```
<ErrorBoundary>
    <EmployeeCard Employee="employee"></EmployeeCard>
</ErrorBoundary>
```

Using Error Boundaries

```
<ErrorBoundary>
    <ChildContent>
        <EmployeeCard Employee="employee"></EmployeeCard>
    </ChildContent>
    <ErrorContent>
        <p>Something went wrong!</p>
    </ErrorContent>
</ErrorBoundary>
```

Showing a Specific Error

Demo



Adding error boundaries

Using Built-in Components

Everything is a component.

Built-in Components

App

Router

DynamicComponent

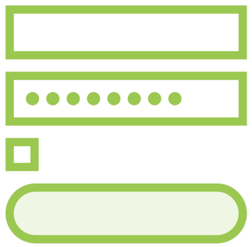
ErrorBoundary

NavMenu

NavLink

Built-in Components

Authentication



Forms

Standalone: PageTitle...

Demo

Setting the title of the page through Blazor

Summary

- Components can handle events and bind data
- Parameters allow components to communicate
- Navigation requires @page and is handled by Router component
- Components can be rendered dynamically
- Error boundaries allow catching errors within a component
- Blazor comes with a set of built-in components

Up next:
Accessing real data using an API