Creating Blazor Components

WRITING YOUR FIRST BLAZOR COMPONENT

Roland Guijt
MICROSOFT MVP, CONSULTANT, AUTHOR AND SPEAKER

@rolandguijt rolandguijt.com

Module Overview

Writing a basic component

Rendering components

Using Razor class libraries

Structuring code

Event Handling

One-way data binding

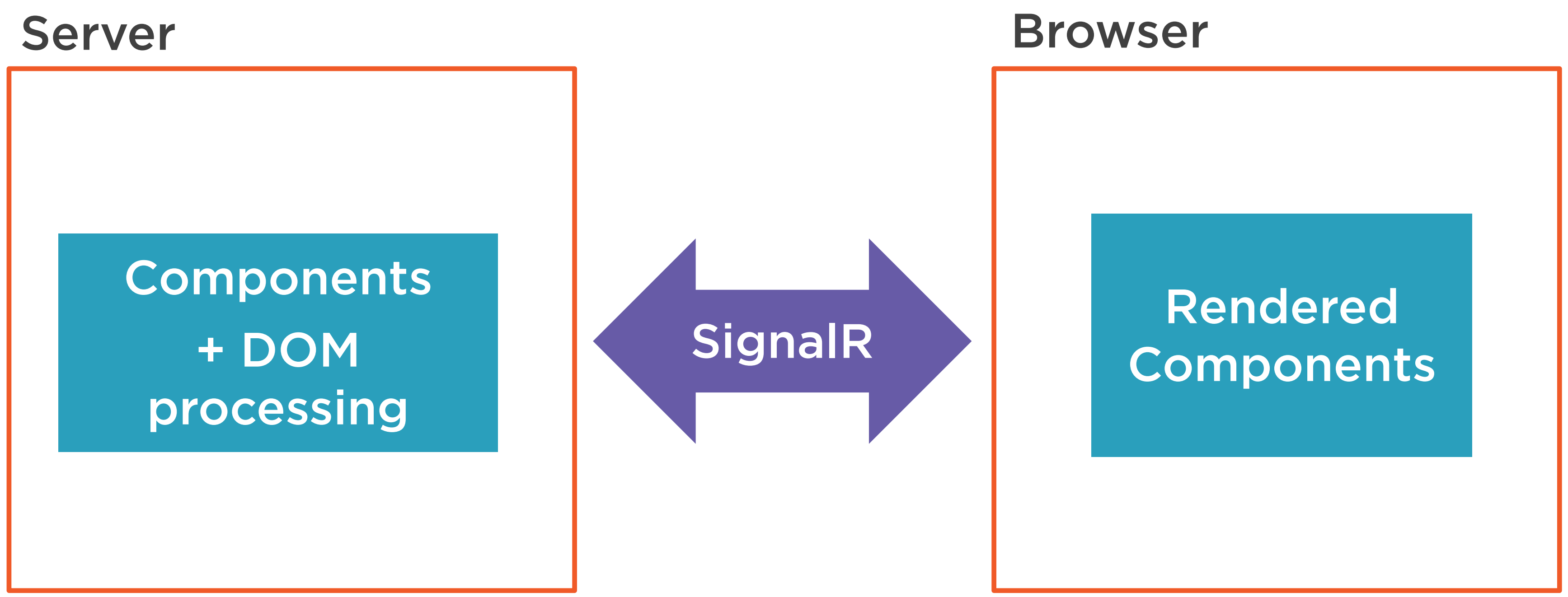
The diff mechanism

Child content

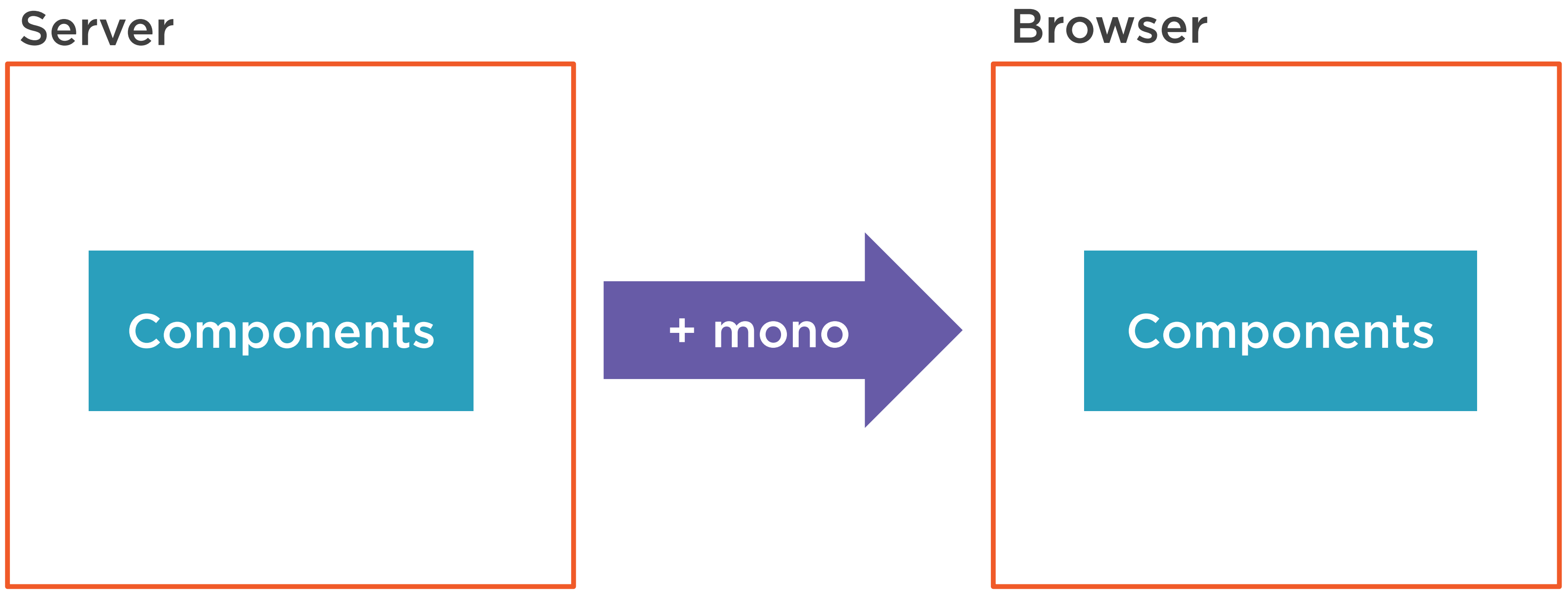
Components use Razor syntax

Components in Components

Hosting Model #1: Blazor Server



Hosting Model #2: Blazor Web Assembly

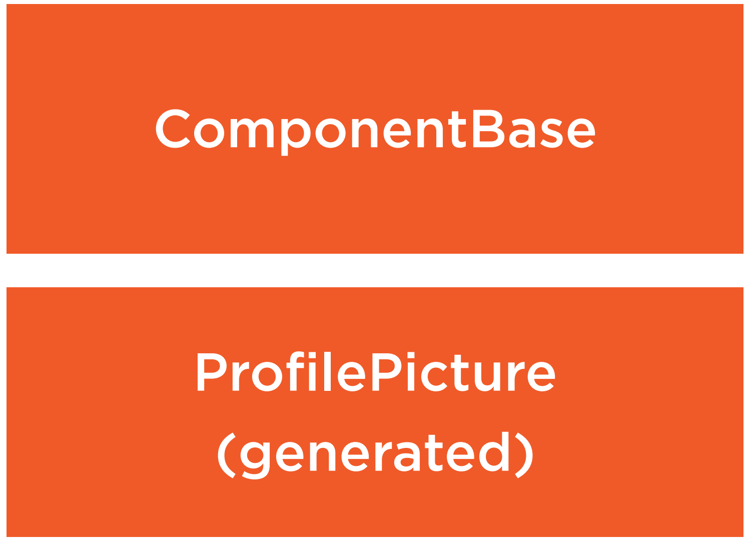


Server and Web Assembly Components

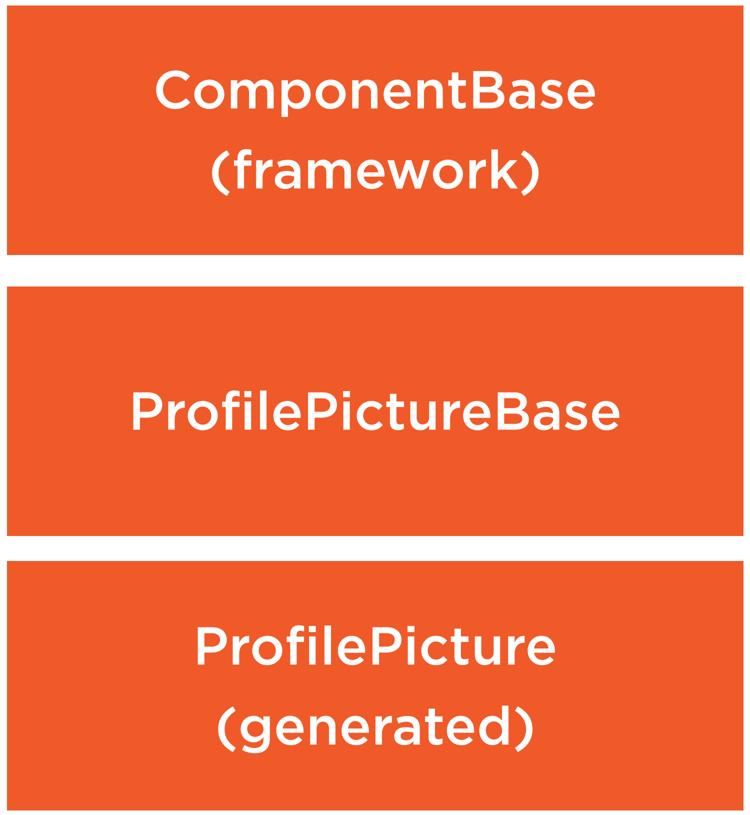
What you can do differs

Structure and features identical

Partial Component Class Hierarchy Without Code-behind



Partial Component Class Hierarchy with Code-behind



ComponentBase
(framework)

BethanysComponentBase

ProfilePictureBase

ProfilePicture
(generated, partial)

Handling Events

```
<img .. @onclick="ProfileClick"/>
```

```
<h2>@message</h2>

@for (var i = 1; i < 4; i++)
{
    var buttonNumber = i;

    <button class="btn btn-primary"
            @onclick="@(e => UpdateHeading(e, buttonNumber))">
        Button #@i
    </button>
}

@code {
    private string message = "Select a button to learn its position.";

    private void UpdateHeading(MouseEventArgs e, int buttonNumber)
    {
        message = $"You selected Button #{buttonNumber} at " +
            $"mouse position: {e.ClientX} X {e.ClientY}.";
    }
}
```

The Diff Mechanism

Old

```
<div>
  <img class = "circle"/>
</div>
```

New

```
<div>
  <img class = ""/>
</div>
```

Update

```
<div>
  <img class = ""/>
</div>
```

Summary

Components are reusable pieces of UI

Anatomy of a component

Handling events

Data binding

Child content

Composing an Interactive Blazor Application

Roland Guijt
MICROSOFT MVP, CONSULTANT, AUTHOR AND SPEAKER

@rolandguijt rolandguijt.com

Module Overview

[Parameter]

Rendering conditionally

Injecting and passing down objects

Lifecycle methods

Two-way data binding

Event callbacks

Form components

Cascading values

Preparation for Benefits Feature

Benefit and EmployeeBenefit entities

Navigation property in Employee entity

EmployeeModel

Added entities to AppDbContext

Migration

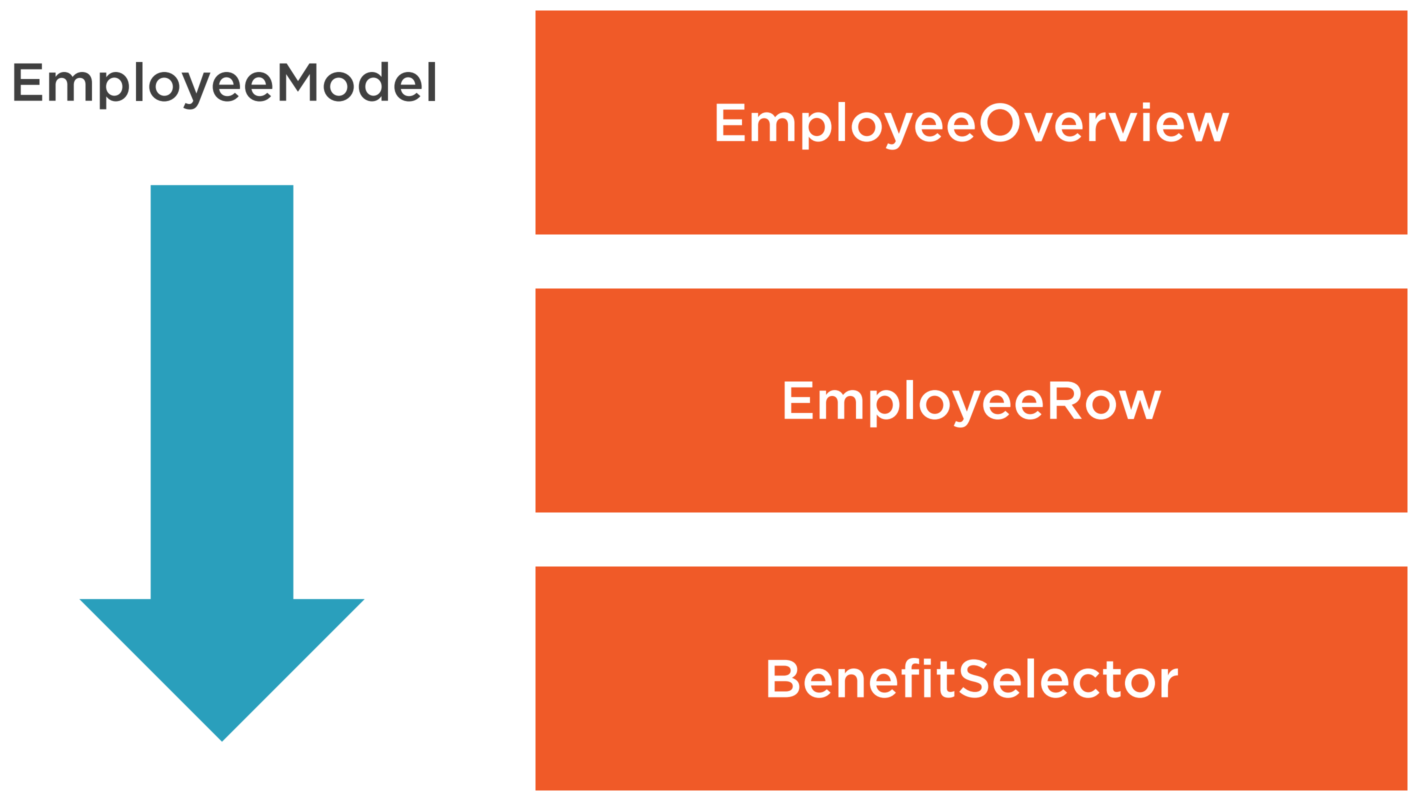
BenefitModel

BenefitRepository and BenefitController

BenefitDataService

https://github.com/GillCleeren
/BethanysPieShopHR

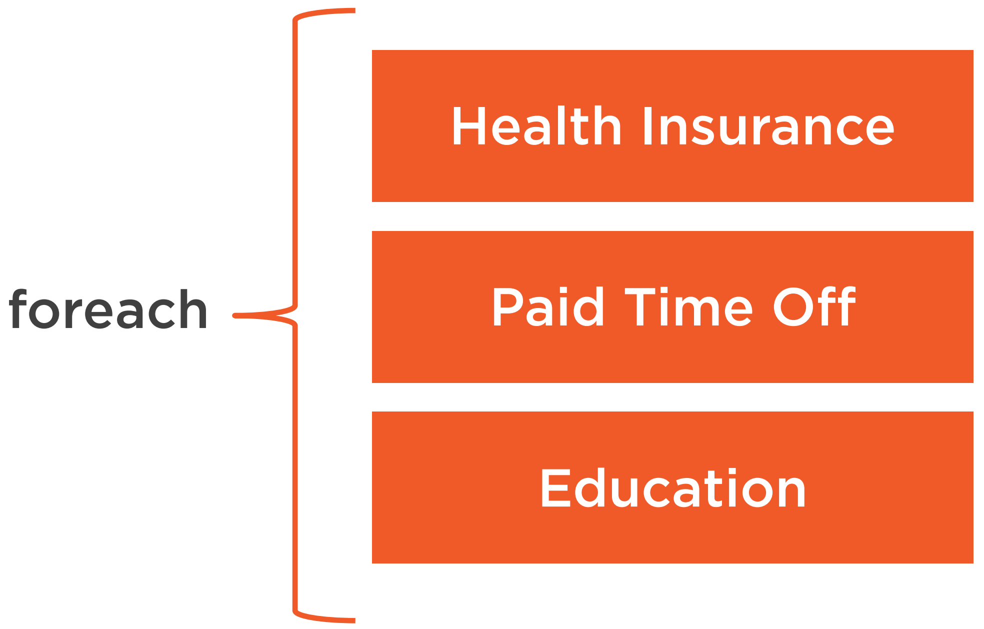
Passing Objects Down the Component Hierarchy

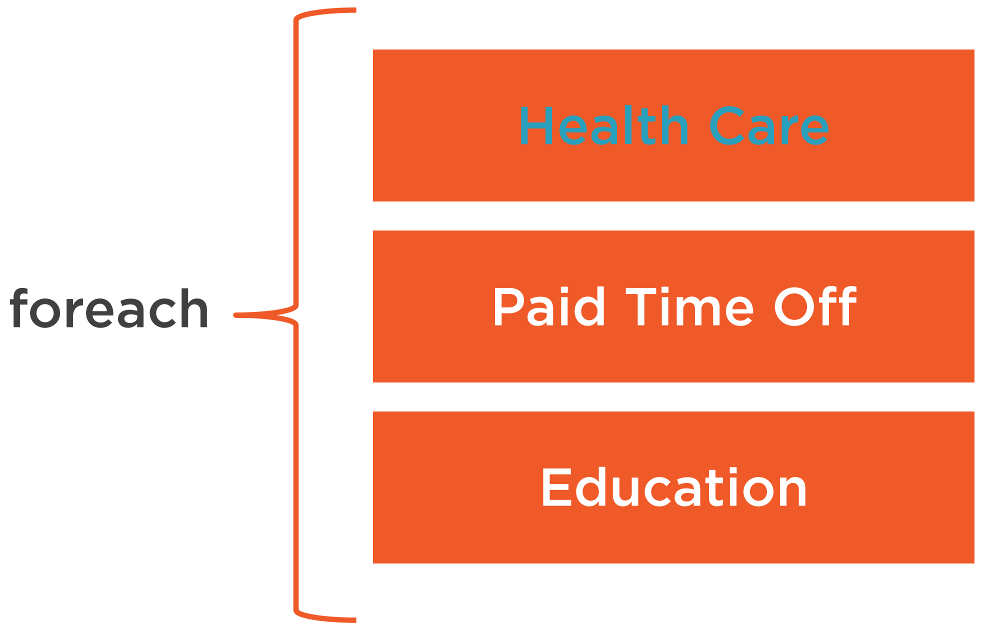


@key

Collection:
Benefit "Health Insurance"
Benefit "Paid Time Off"
Benefit "Education"

Collection:
Benefit "Health Care"
Benefit "Paid Time Off"
Benefit "Education"

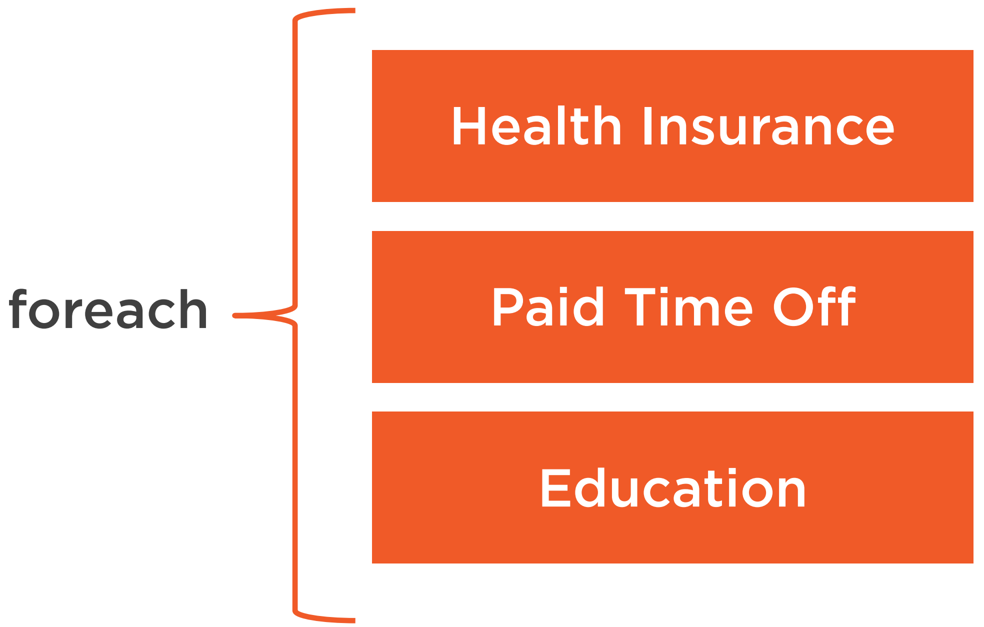


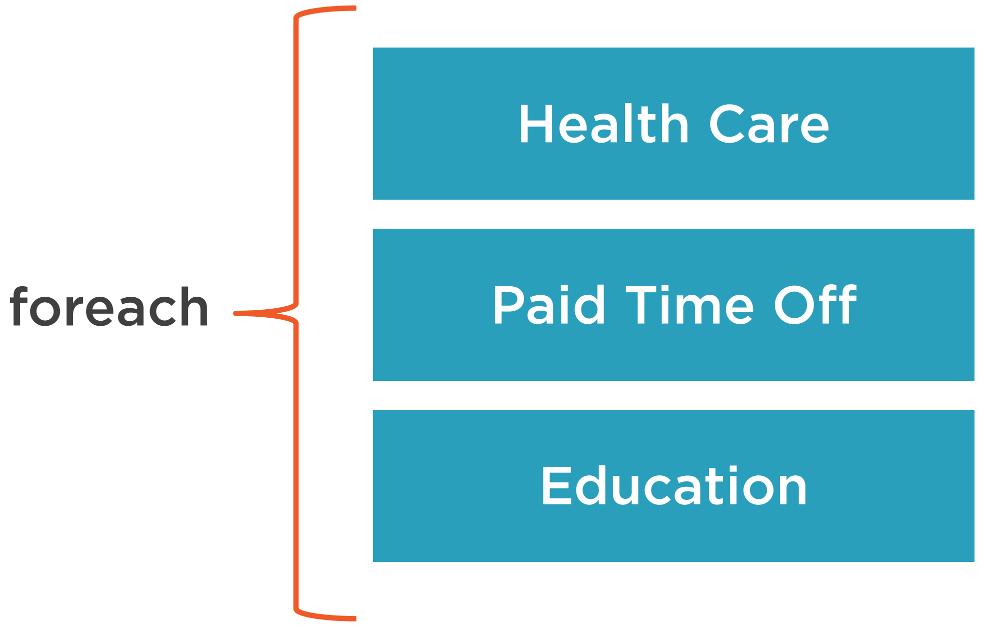


@key

Collection:
Benefit "Health Insurance"
Benefit "Paid Time Off"
Benefit "Education"

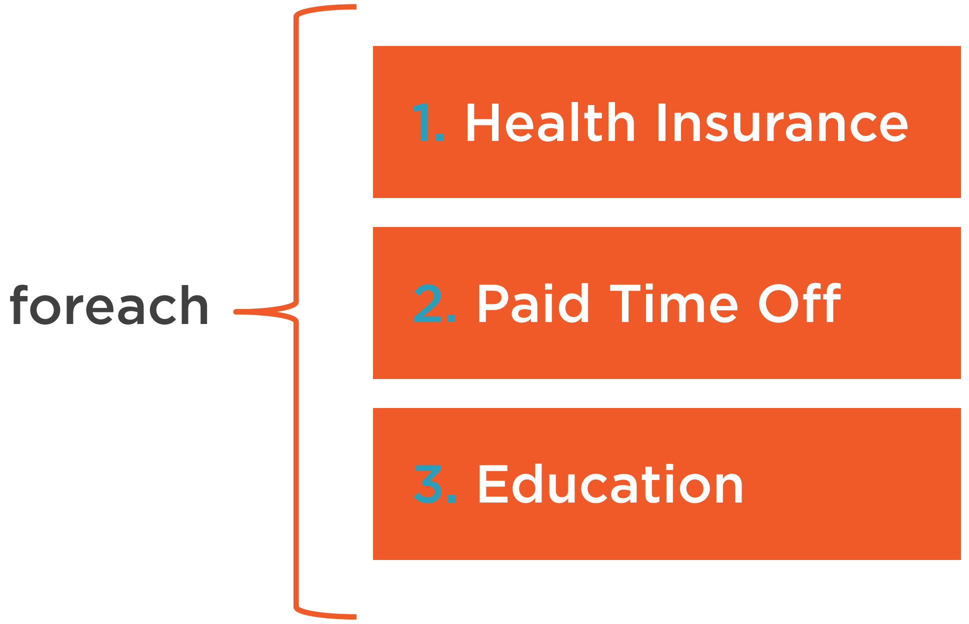
Collection:
Benefit "Health Care"
Benefit "Paid Time Off"
Benefit "Education"





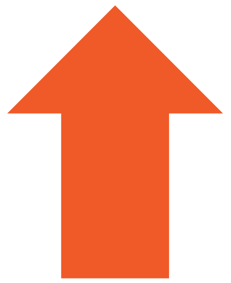
@key

Collection:
Benefit 1 "Health Insurance"
Benefit 2 "Paid Time Off"
Benefit 3 "Education"



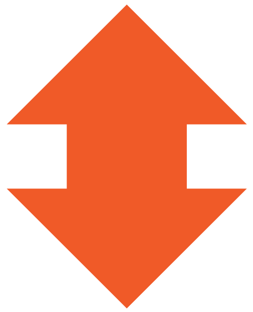
Two-way Data Binding

```
<img class = "@cssClass" />
```



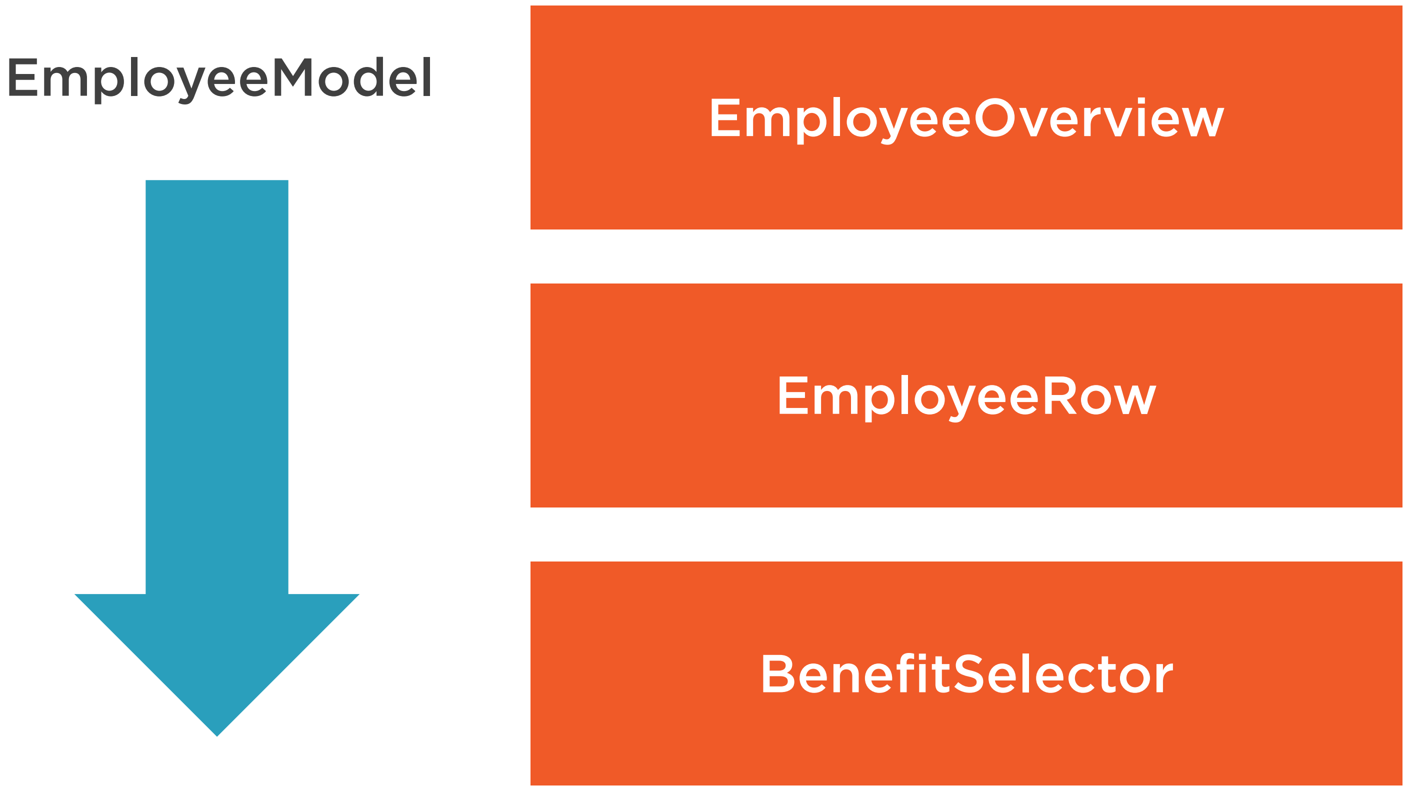
```
private string cssClass = "circle";
```

```
<input checked = "@Selected" />
```

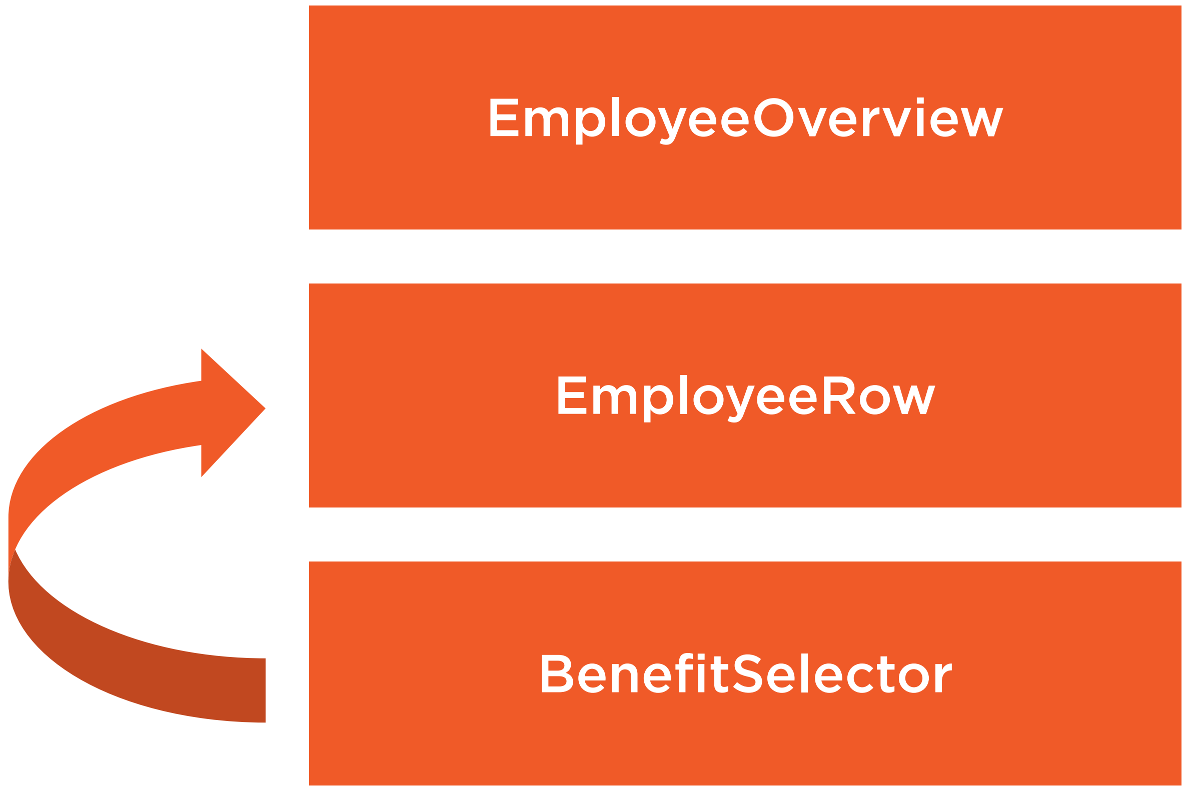


```
public bool Selected { get; set; }
```

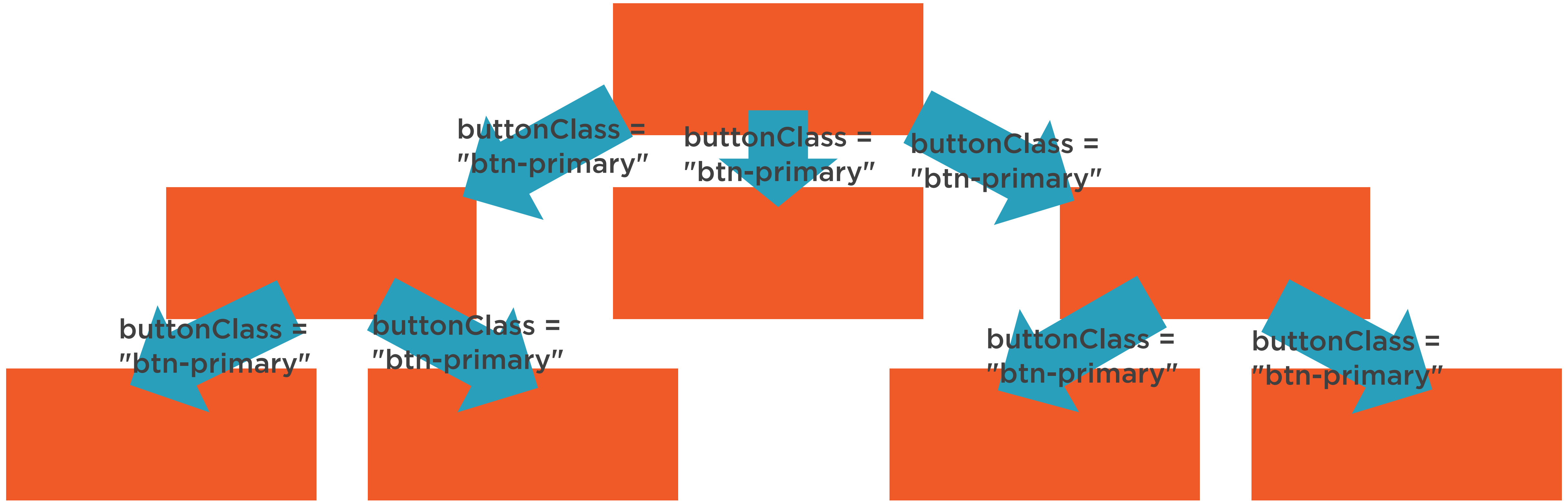
Component Hierarchy



Component Hierarchy



Cascading Values


buttonClass = "btn-primary"
buttonClass = "btn-primary"
buttonClass = "btn-primary"
buttonClass = "btn-primary"
buttonClass = "btn-primary"
buttonClass = "btn-primary"
buttonClass = "btn-primary"

Cascading Values

```
<CascadingValue Value="@buttonClass">
        <CascadingValue Value="@inputClass">
                <EmployeeOverView>
                        <AddEmployeeDialog>
                        [CascadingParameter]
                        public string CascadingValue { get; set ; }
```

Cascading Values

```
<CascadingValue Value="@buttonClass" Name="button">
    <CascadingValue Value="@inputClass" Name="input">
        <EmployeeOverView>
            <AddEmployeeDialog>
            [CascadingParameter(Name = "button")]
            public string CascadingValue { get; set ; }
```

Summary

Component interaction

The lifecycle

Two-way data binding

Creating Templated Components

Roland Guijt
MICROSOFT MVP, CONSULTANT, AUTHOR AND SPEAKER

@rolandguijt rolandguijt.com

Module Overview

Razor template syntax

Creating a templated component

Consuming templated components

Applying generics to templates

Summary

Razor templates++

Templates are about reusability of components

Generics == reusability++

Partial class enables generic where clause