Developing Cross-Platform Web Apps With Blazor

Wael Kdouh - @waelkdouh

Senior Customer Engineer

v1.0

How to View This Presentation

- To switch to **Notes Page** view:
  - On the ribbon, click the **View** tab, and then click **Notes Page**

- To navigate through notes, use the Page Up and Page Down keys
  - Zoom in or zoom out, if required

- In the **Notes Page** view, you can:
  - Read any supporting text—now or after the delivery
  - Add notes to your copy of the presentation, if required

- Take the presentation files home with you

Module 7: Forms and Validation

Module Overview

Module 7: Forms and Validation

Section 1: Forms

Lesson: Overview

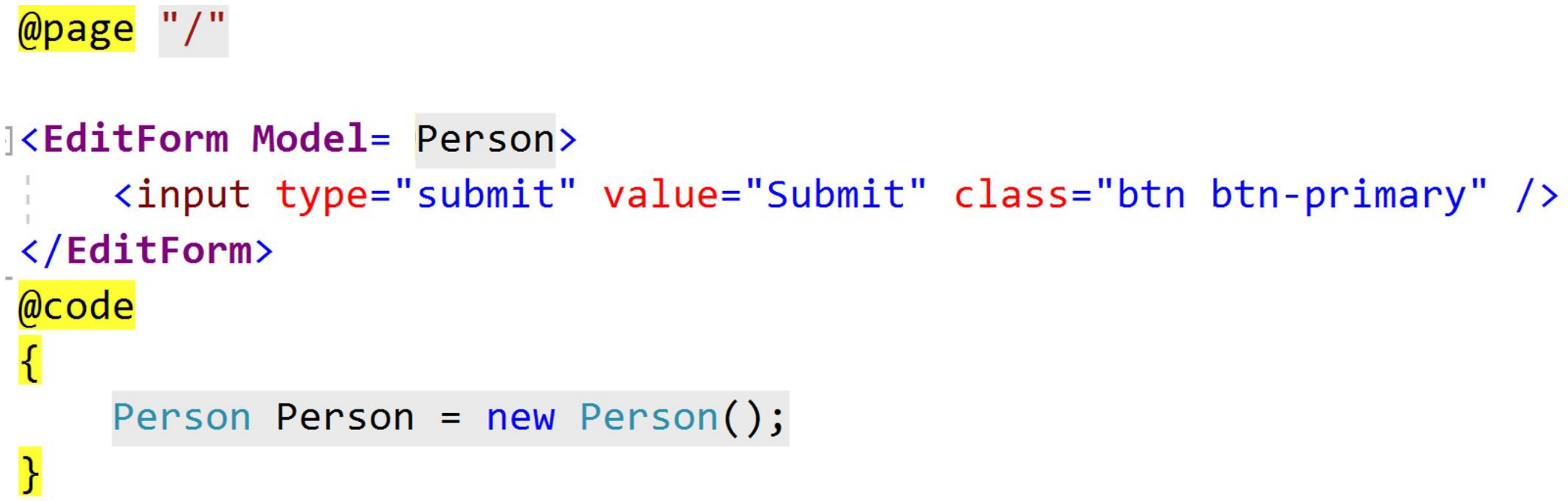
Forms

- The EditForm component is Blazor’s approach to managing user-input in a way that makes it easy to perform validation and represent validity state to the user

- Although it is possible to create forms using the standard <form> HTML element, its recommend to use the EditForm component because of the additional features it provides

The Form Model

- **The key feature to the EditForm is its Model parameter**. This parameter provides the component with a context it can work with to enable user-interface binding and determine whether or not the user's input is valid

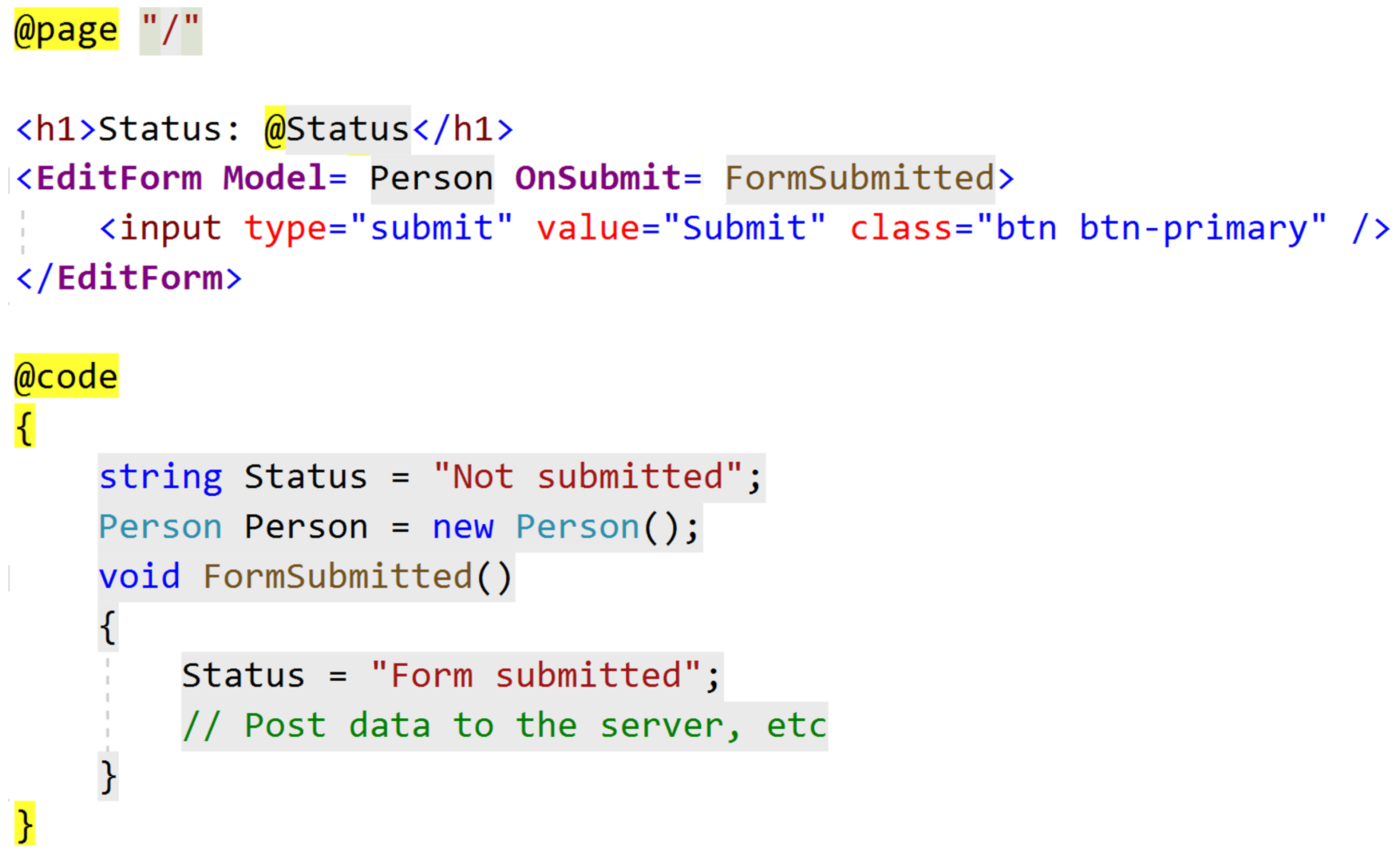


```
@page "/"

<EditForm Model= Person>
    <input type="submit" value="Submit" class="btn btn-primary" />
</EditForm>
@code
{
    Person Person = new Person();
}
```

Detecting Form Submission

- When the **user clicks the Submit button, the EditForm will trigger its OnSubmit** event. You can use this event in the code to handle any business logic



```
@page "/"

<h1>Status: @Status</h1>
<EditForm Model= Person OnSubmit= FormSubmitted>
    <input type="submit" value="Submit" class="btn btn-primary" />
</EditForm>

@code
{
    string Status = "Not submitted";
    Person Person = new Person();
    void FormSubmitted()
    {
        Status = "Form submitted";
        // Post data to the server, etc
    }
}
```

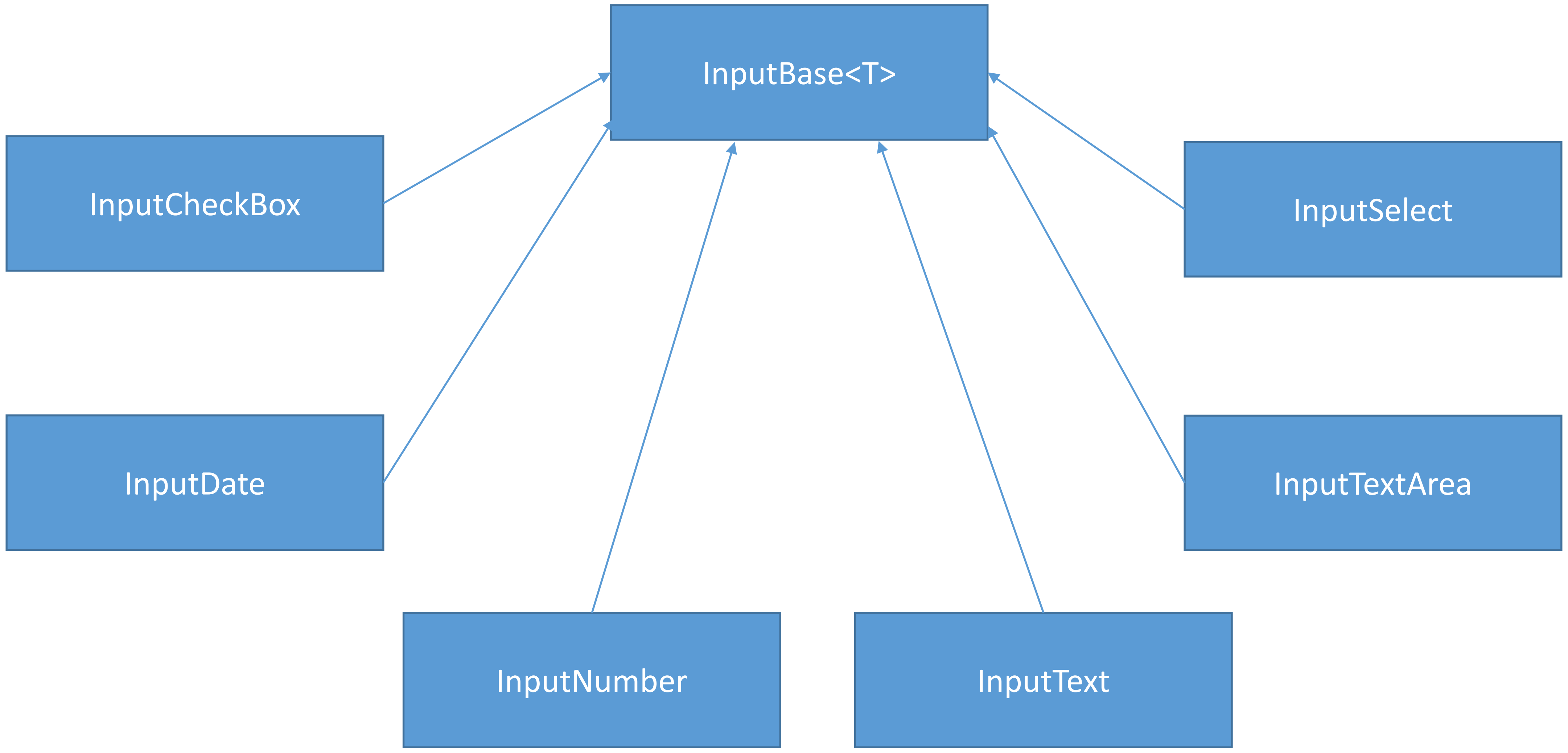
Editing Form Data

- Because the EditForm component renders a standard <form> HTML element, it is actually possible to use standard HTML form elements such as <input> and <select> within our mark-up

- But as with the EditForm component it is recommended using the various **Blazor input controls,** because they **come with additional functionality such as validation**

- There is a **standard collection of input components** available in Blazor, all **descended from the base class InputBase<T>**

Editing Form Data



InputCheckbox

- The InputCheckbox component binds a Boolean property to an HTML <input> element with type="checkbox". This component does not allow binding to a nullable property

- **<InputCheckbox @bind-Value=FormData.SomeBooleanProperty />**

InputDate

- The InputDate components binds a DateTime property to an HTML <input> element with type="date". This component will bind to a nullable property, however, not all browsers provide the ability to specify a null value on an input element of this type

- **<InputDate @bind-Value=FormData.SomeDateTimeProperty ParsingErrorMessage="Must be a date" />**

InputNumber

- The InputNumber component binds any kind of C# numerical property to an HTML <input> element with type="number"
- If the value entered cannot be parsed into the target property type the input will be considered invalid and will not update the Model with the value
- When the target property is nullable, an invalid input will be considered null and the text in the input will be cleared
- **<InputNumber @bind-Value=FormData.SomeIntegerProperty ParsingErrorMessage="Must be an integer value" />**
- **<InputNumber @bind-Value=FormData.SomeDecimalProperty ParsingErrorMessage="Must be a decimal value" />**

InputText

- The InputText components binds a string property to an HTML <input> element with no type specified. This enables specifying any of the available input types such as password, color, or one of the other options as specified in the W3 standards

- **<InputText @bind-Value=FormData.SomeStringProperty />**

InputTextArea

- The InputTextArea components binds a string property to an HTML <textarea> element

- **<InputTextArea @bind-Value=FormData.SomeMultiLineStringProperty />**

InputSelect

- The InputSelect component binds a property of any kind to an HTML <select> element. Blazor will automatically select the correct <option> based on the value of the property

- **<InputSelect @bind-Value=FormData.SomeSelectProperty>**
  <option value="Pending">Pending</option>
  <option value="Active">Active</option>
  <option value="Suspended">Suspended</option>
  </InputSelect>

Demo: Forms

Module 7: Forms and Validation

Section 2: Validation

Lesson: Overview

Validation

- The **DataAnnotationsValidator is the standard validator type in Blazor**

- Adding this component within an EditForm component will enable form validation based on .NET attributes descended from System.ComponentModel.DataAnnotations.ValidationAttribute

Displaying Validation Error Messages

- Validation **error messages can be displayed to the user in two ways**
  - Add a **ValidationSummary** to show a comprehensive **list of all errors** in the form
  - Use the **ValidationMessage** component to display **error messages for a specific input** on the form

- These components are not mutually exclusive, so it is possible to use both at the same time

ValidationSummary

- The ValidationSummary component can simply be dropped into an EditForm into the mark-up; no additional parameters are required at all

ValidationMessage

- As the ValidationMessage component displays error messages for a single field, it requires specifying the identity of the field
- To ensure that the parameter's value is never incorrect (even when refactoring property names on the Person class) Blazor requires specifying an Expression when identifying the field
- The parameter, named For, is defined on the ValidationMessage as follows:

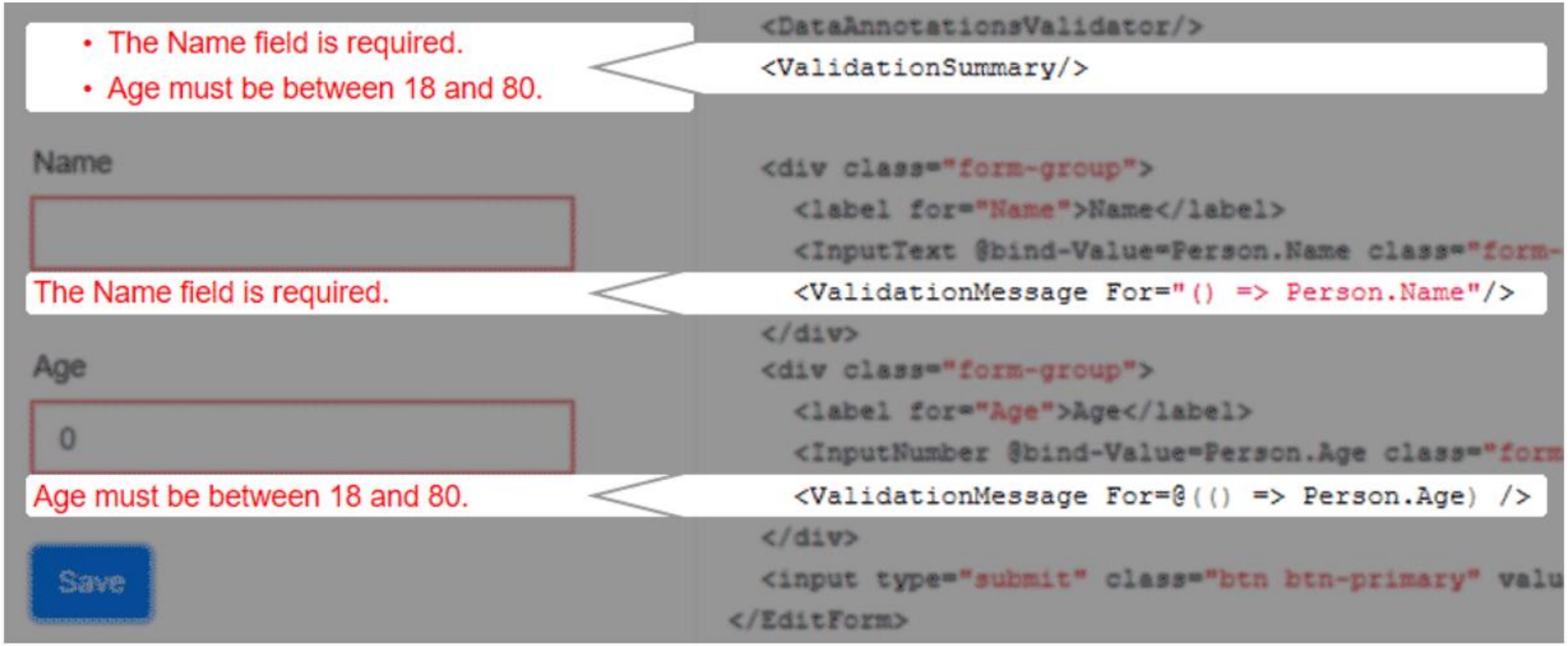
```
[Parameter]
public Expression<Func<T>> For { get; set; }
```

ValidationMessage

- This means to specify the identity of the field you should use a lambda expression, which can be presented either “quoted”, or wrapped in @(...)
- Quoted form
  - <ValidationMessage For="() => Person.Name"/>
- Razor expression form
  - <ValidationMessage For=@(() => Person.Name)/>

- Both forms are equivalent. The quoted form is easier to read, whereas the razor expression makes it more obvious to other developers that you are defining an expression rather than a string

ValidationMessage



- The Name field is required.
- Age must be between 18 and 80.

Name

The Name field is required.

Age

0

Age must be between 18 and 80.

Save

```
  <DataAnnotationsValidator/>
  <ValidationSummary/>

  <div class="form-group">
    <label for="Name">Name</label>
    <InputText @bind-Value=Person.Name class="form-
    <ValidationMessage For="() => Person.Name"/>
  </div>
  <div class="form-group">
    <label for="Age">Age</label>
    <InputNumber @bind-Value=Person.Age class="form
    <ValidationMessage For=@(() => Person.Age) />
  </div>
  <input type="submit" class="btn btn-primary" valu
</EditForm>
```

Demo: Validation

Module 7: Forms and Validation

Section 2: Validation

Lesson: Handling Form Submission

Handling Form Submission

- When rendering an EditForm component, Blazor will output an HTML <form> element

- Since this is a standard web control, you can provide the user with the ability to submit the form by adding an <input> with type="submit“

- Blazor will intercept form submission events and route them back through to the razor view. There are three events on an EditForm related to form submission:
  - OnValidSubmit
  - OnInvalidSubmit
  - OnSubmit

- **Each of these events pass an EditContext as a parameter**, which you can use to determine the status of the user’s input

Handling Form Submission

- You can use none of these events or one of these events
  - OnValidSubmit
  - OnInvalidSubmit
  - OnSubmit

- The only situation where you can use two events is when you set OnValidSubmit and OnInvalidSubmit together. Neither of those two events can be consumed if OnSubmit is set

OnValidSubmit / OnInvalidSubmit

- The OnValidSubmit event is executed when the form passes validation

- The OnInvalidSubmit event is executed when the form fails validation

OnSubmit

- The OnSubmit event is executed when the form is submitted, regardless of whether the form passes validation or not

- It is possible to check the validity status of the form by executing editContext.Validate(), which returns true if the form is valid or false if it is invalid (has validation errors)

Enable The Submit Button Based On Form Validation

- To enable and disable the submit button based on form validation:
  - Use the form's EditContext to assign the model when the component is initialized
  - Validate the form in the context's OnFieldChanged callback to enable and disable the submit button
  - Unhook the event handler in the Dispose method

- Note: Model parameter is not used when explicitly passing the EditContext

Demo: Handling Form Submission

Custom Validation

- There are a lot of validation attributes provided with the Annotations library, but sometimes a new rule emerges that is not supported

- For these kinds of rules, we must create a custom attribute and apply it to our model class

- Steps to create custom validation
  - Create a class that inherits from the ValidationAttribute abstract class, which is the base class for the validation attributes
  - Create properties which we can set from the model class when calling this attribute
  - After the property creation, we have to override the IsValid method that accepts two parameters. The value parameter will hold the value the user enters in the input field. The validationContext parameter describes the context in which we perform the validation

Demo: Dynamic Form Validation

Module Summary

- In this module, you learned about:
  - Forms
  - Editing Forms Data
  - Validation
  - Handling Form Submission

Lab 7: Forms and Validation

References

- Microsoft Docs

- Blazor University

Microsoft