Blazor Basics

What is it, how does it work, when and how to use.

Henk Holterman
h.holterman@4dotnet.nl

Agenda: Day 2

- Summary of Day 1
- Overflow Day 1
- Testing with bUnit, testing the TabControl
- A deep look at Databinding
- Blazor Syntax
- A simple MD editor with Preview
- StateManagement
- JS Interop (GeoLocation)

Blazor Rendermodes

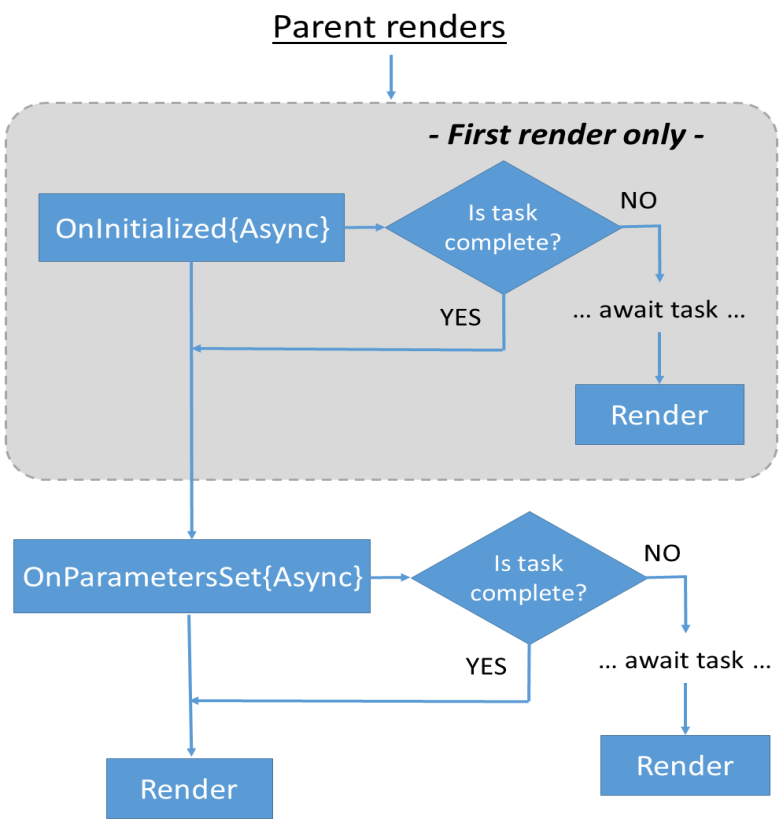
Rendermode	Drag & Drop	SEO	Scalability	Offline	Effort
Static Serverside	No	Good	Good	No	No API: ± 30% less work
Serverside Interactive	Yes	requires Prerendering	Medium	No	
WebAssembly	Yes	requires Prerendering	Very Good	Yes, also PWA	Requires an API

- *Serverside Interactive* ticks a lot of boxes but it requires good internet connections and a lot of RAM on the server.
  - A rough guideline: up to a [few] thousand concurrent users will work well.
  - Mixing with static pages might improve scalability.
  - There is no built-in support for scale-out.

The Event Lifecycle, pt 1: Initialization

- OnInitialized (1 time) and OnParametersSet (1+ times) are the two main init events.
- They both have ...Async versions
- When you use an **`await`** then a Render may happen *before* OnInitializedAsync is finished.

- Look at **Weather.razor** and the null handling there. Is it necessary?

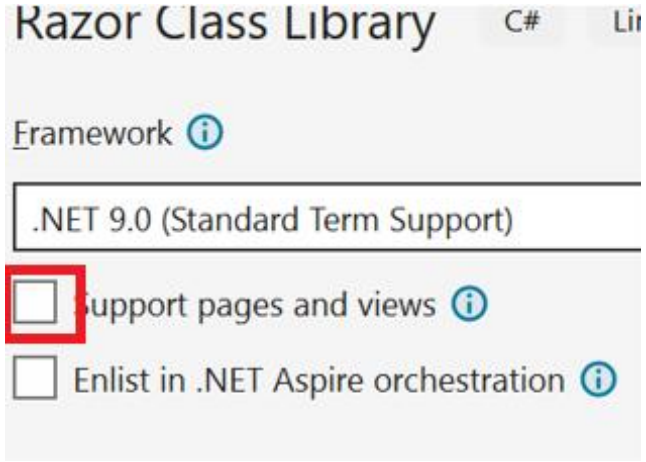


Fixing the Clock

- The GUI (virtual DOM) is not thread-safe.
  - Fix 1: `InvokeAsync(StateHasChanged);`
  - Run. No errors, but is it ticking?
- By default the Home page is not interactive.
  - Fix 2: add `@rendermode InteractiveServer` on top
- This component would still leak resources
  - Fix 3: add
    `@implements IDisposable` (on top)
    `public void Dispose() => timer?.Dispose();` (in @code)

Using a Razor Class Library

- Create a new Web App project, call it RclDemo
- Add a Razor Class Library to the Solution, name it MyComponents
  - Note: do not check that box. We want a blazor library, not a Razor Pages one.
- Add a Project reference from RclDemo to MyComponents
- And then “it just works”



TabControls, rounding up

- Now we can finish the controls:

In TabControl

```
<CascadingValue …>
<ul class="nav nav-tabs">
    @ChildContent
</ul>
</CascadingValue>
<div class="nav-tabs-body">
    @ActivePage?.ChildContent
</div>
```

In TabPage

```
<li><a class="nav-link @IsActiveClass"
       role="button"
       @onclick="ShowMe">
    @Title</a></li>

string IsActiveClass =>
  Parent.IsActivePage(this) ? "active"  : "";
void ShowMe() => Parent.SetActivePage(this);
```

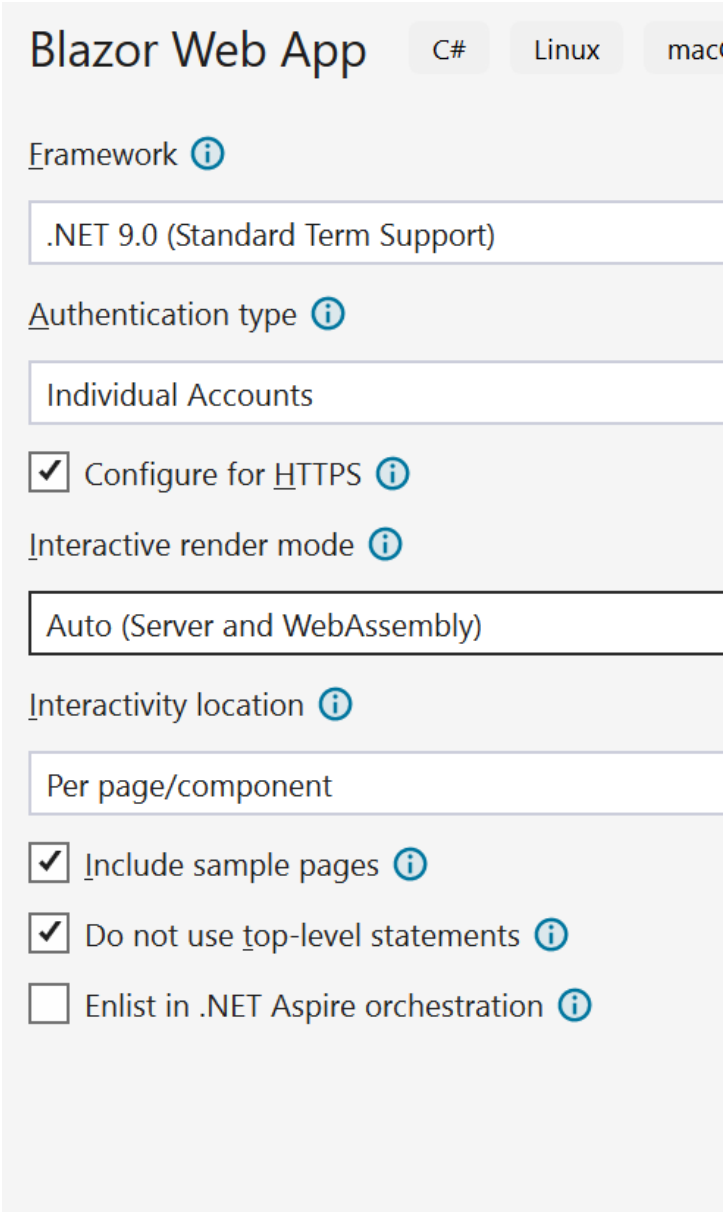
The other Templates

- Create a new Blazor Web App, settings →
- Create an Info.razor:

```
<div> Render: @RendererInfo.Name
 (@RendererInfo.IsInteractive)
</div>
```

- Put an <Info /> on all pages we look at.
- The Account ‘Area’ has own Layout etc.

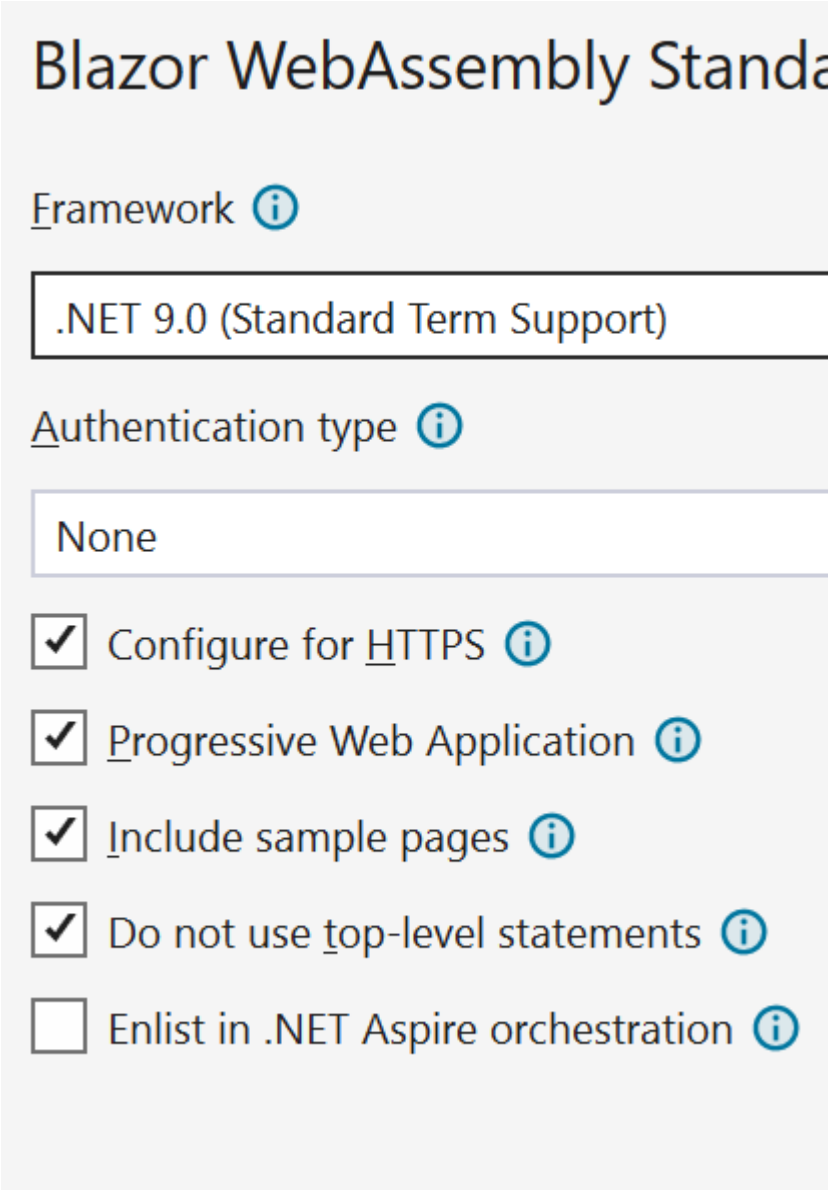
- Discussion, uses-cases?



A look at Wasm and PWA

- Create a new WebAssembly App →
- Add the same Info component

- PWA only ‘works’ after publishing



Templated components

- Create a new project, name it TemplatesDemo
- Look at the Weather page, especially the table.
- Use-case: we will have many such tables and want to make it more reusable.
- Add a TableComponent.razor, copy/paste the <table> into it.
- In the TableComponent, remove everything inside the <tr> tags. Keep the @foreach.
- In the @code section, add:

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

[Parameter, EditorRequired]
public RenderFragment<T> RowTemplate { get; set; }

[Parameter, EditorRequired]
public IEnumerable<T>? Items { get; set; }
```

- Use the RenderFragments inside the <tr> tags.

Templated components (2)

- At the top of TableComponent.razor, add

```
@typeparam T
```

- In Weather.razor, add

```
<TableComponent Items="forecasts">
  <HeaderTemplate>
     copy/paste the contents of the <thead><tr> here
  </HeaderTemplate>
  <RowTemplate Context="forecast">
     copy/paste the contents of the <tbody><tr> here
  </RowTemplate>
</TableComponent>
```

- Then delete the original table, Run

Templated components (3)

- Watch: another way to supply the HeaderTemplate. ChildContent is just the default RenderFragment name.

- Exercise: add an extra column inside the TableComponent. Add 2 buttons, Edit and Delete to each row. Do not change the Weather page.

TabControl extra: use a component from code

- Suppose we want to show a certain page from the Home page logic.
- Add 2 variables, `TabControl mainTab;` and `TabPage weather;`
- Add an `@ref` element: `<TabControl @ref="mainTabs">`

- Add a Button to the Home page to do:

```
mainTab.SetActivePage(weather);
```

- Note: @ref assignments are handled during a render, so they are not available in OnInitialized.

Testing: bUnit intro

- Create a new Project, name it TestDemo
- Add an Xunit Testproject to the solution, eg TestDemoTests and reference the TestDemo project
- Add the Bunit packet from NuGet
- Code (from learn.microsoft.com/en-us/aspnet/core/blazor/test)

```
[Fact]
public void CounterShouldIncrementWhenClicked()
{
  // Arrange
  using var ctx = new TestContext();
  var cut = ctx.RenderComponent<Counter>();
  var paraElm = cut.Find("p");

  // Act
  cut.Find("button").Click();

  // Assert
  var paraElmText = paraElm.TextContent;
  paraElmText.MarkupMatches("Current count: 1");
}
```

A little bit more real: testing the TabControl

- Open the TabControls project (maybe make a copy)
- Add a Test project, add the BUnit package and the project reference
- Add `TabControlTests.razor` and _Imports.razor
- In a .razor file we can use Razor, add an @code section and ten the following:

```
RenderFragment fixture =>
  @<TabControl>
    <TabPage Title="1"><p>één</p></TabPage>
    <TabPage Title="2"><p>twee</p></TabPage>
  </TabControl> ;
```

Test outline:

```
[Fact]
public void TabControlShouldActivatePageOnClick()
{
    // Arrange
    using var ctx = new TestContext();
    var subject = ctx.Render(fixture);

    // act
    var heading2 = subject.FindAll("a").Last();
    heading2.Click();

    // Assert
    var pageContent = subject.Find("p");

    pageContent.MarkupMatches(@<p>two</p>);
}
```

- Note: Try, but it won’t work yet …

Making it work

- The code so far looks Ok and Builds Ok but the Tests won’t run
- Open both Project files (XML)
- Compare the first line.
- Add .Web to get `<Project Sdk="Microsoft.NET.Sdk.Web">`
- VS will ask to reload the project. Then run the Test again.

- Think of something else to Test.

Databinding in Blazor: 1-way binding

- Create a new Project, name it BoundCounter
- Make the Home page interactive and add

```
<Counter Step="5" />
```

- Change Counter to make this work.

Databinding in Blazor: 2-way binding

- In Home.razor, add

```
<p>Home count: @myCount</p>
<Counter Step="10" Count="myCount"/>
@code {
    int myCount = 100;
}
```

- We want this Counter to start at 100 and update myCount.

Databinding in Blazor: 2-way binding

- First add a 1-way binding for Count to Counter.razor.
- Inside, replace currentCount with Count
- Test this. It should start at 100 but not update myCount.
- Add an EventCallback, this has to follow a naming pattern:

```
[Parameter]
public EventCallback<int> CountChanged { get; set; }
private void IncrementCount()
{
  CountChanged.InvokeAsync(Count + Step);
}
```

- Run this, the counter now does nothing.

Databinding in Blazor: 2-way binding

- In Home.razor, add this:

```
<Counter Step="10" Count="myCount" CountChanged="c => myCount = c"/>
```

- The Counter should now do what we want.

- This is the basic pattern, used a lot. There is a shorthand notation. Add a second Counter like this:

```
<Counter Step="5" @bind-Count="myCount"  />
```

- This does exactly the same and both Counters bind to the same variable.

Blazor Syntax: parameters and quotes

MS: Quotes around parameter attribute values are optional in most cases per the HTML5 specification. For example, Value=this is supported, instead of Value="this". However, we recommend using quotes because it's easier to remember and widely adopted across web-based technologies.

```
<MyComponent X=1  Y="2"  Z='3' />
```

- All three will work but X=1 is disadvised.

Blazor syntax: the @ prefix

MS: If the component parameter is a variable of type string, then the attribute value is instead treated as a C# string literal by default. If you want to specify a C# expression instead, then use the @ prefix.

`Message="message"` and `Message="@message"` are different.

`Size="@size"` and `Size="size"` are exactly equal

Blazor syntax: escaping the @

```
string name = "john", domain = "doe.com";
<p>@name@@@domain</p>  // becomes  john@doe.com
```

And string interpolation ($) looks like this:

```
Message="@($"{name}")"
```

Blazor syntax, parameters and directives

`@onclick="IncrementCount"`	Ok
`onclick="IncrementCount"`	JS error

- The thing on the right hand side of = is called a *value*.
- On the left we have:

- HTML Attributes	class="btn" onclick="click"
- Blazor Parameters	Message="@message" Rows="12"
- Blazor Directives	@bind="Value" @onclick="Click"

Miscellaneous: the for loop

- Create a new Project and add:

```
@for (int i = 0; i < items.Count; i++)
{
  <div >
    @i: @items[i]
     <button @onclick="() => Delete(i)">X</button>
  </div>
}
```

- Add `@code` to set up `items` and `void Delete(int index)`.
- Run and try to delete an item. Expect an exception.

Build a MD editor with a Preview

- New Project, add an Editor page, add

```
<textarea rows="10" cols="50" @bind="@mdText"
@bind:event="oninput" />
```

- Add a NuGet package for MD, eg Markdig
- Convert to a string and display in the page.
  How to show it in real-time?
  Why does this show HTML?
- Blazor has a MarkupString to display ‘unsafe’ text.

Browser Storage

Soort	Algemeen	Blazor Server	Blazor Wasm
Session	Scope: 1 Browser Tab	Size in kB. Can encrypt.	Limit 5-10 MB.
Local	Scope: All Tabs, persistent	Size in kB. Can encrypt.	Limit 5-10 MB.
IndexedDb	Semi SQL	Not relevant	Limit 10 MB – 2GB
Browser Cache	Linked to URL	Not relevant	Used by Blazor, PWA

- Blazor Server Interactive:
  - transport over SignalR. Not suited for large data.
  - The server can encrypt, hiding the content from the user.

- The limits (quota) are Browser dependent, this are averages.

Limits of razor: creating html tags

- The challenge: can we make a <h3> tag like this?

```
<Header Level="3">
    Im a header tag
</Header>
```

1. Try to make a Header1.razor,
   see if you can do this.
2. Make a Header2.cs,
   - inherit from ComponentBase
   - override **BuildRenderTree**

Why? This is an entirely different way to do Blazor, sometimes you will need it.

Short intro JS-Interop: GeoLocation

- This is prepared code, we look at it together.

Questions, repeats ?

Agenda: About Day 3

- Some lose ends, tips and tricks
- Build a simple Line-of-Business application
  - A prepared model & database
  - Building some List and Edit forms
  - Add validation
- Options
  - Localization
  - Scaffolding
  - … suggestions ?