Blazor Basics

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

Henk Holterman
h.holterman@4dotnet.nl

Introductions

- A short intro round
  - Your name
  - Experience with Web
  - Expectations

- Quick installation check, we will need
  - VS 2022 and dotnet 9
  - with the “ASP.NET and web development” workload
  - For tomorrow: a github username

Agenda: Day 1

- Blazor in the web development landscape
- A short Blazor history
- Blazor Rendermodes
  - A quick look at the starter templates
- The Event Lifecycle
- Creating a simple component
- Creating a TabControl/TabPage combo
- Adding components to a Razor Class Library (RCL)

Blazor compared

Name	Http	Popular Frameworks	Database	Blazor
App/Site (MPA)	GET, POST	PHP, MVC, Razor pages	direct	Static Serverside
	1 x GET, WebSockets		direct	Serverside Interactive
Single Page: SPA	1 x GET	Angular, React, VUE	via API	WebAssembly
API	GET, POST, DELETE, …	ASP.NET	direct	

- SPA: Single Page App, Google Maps led the way.
- MPA: traditional Request/Response apps

WebAssembly

http://webassembly.org/

Binary instruction format
for a stack-based VM
For Browser and beyond

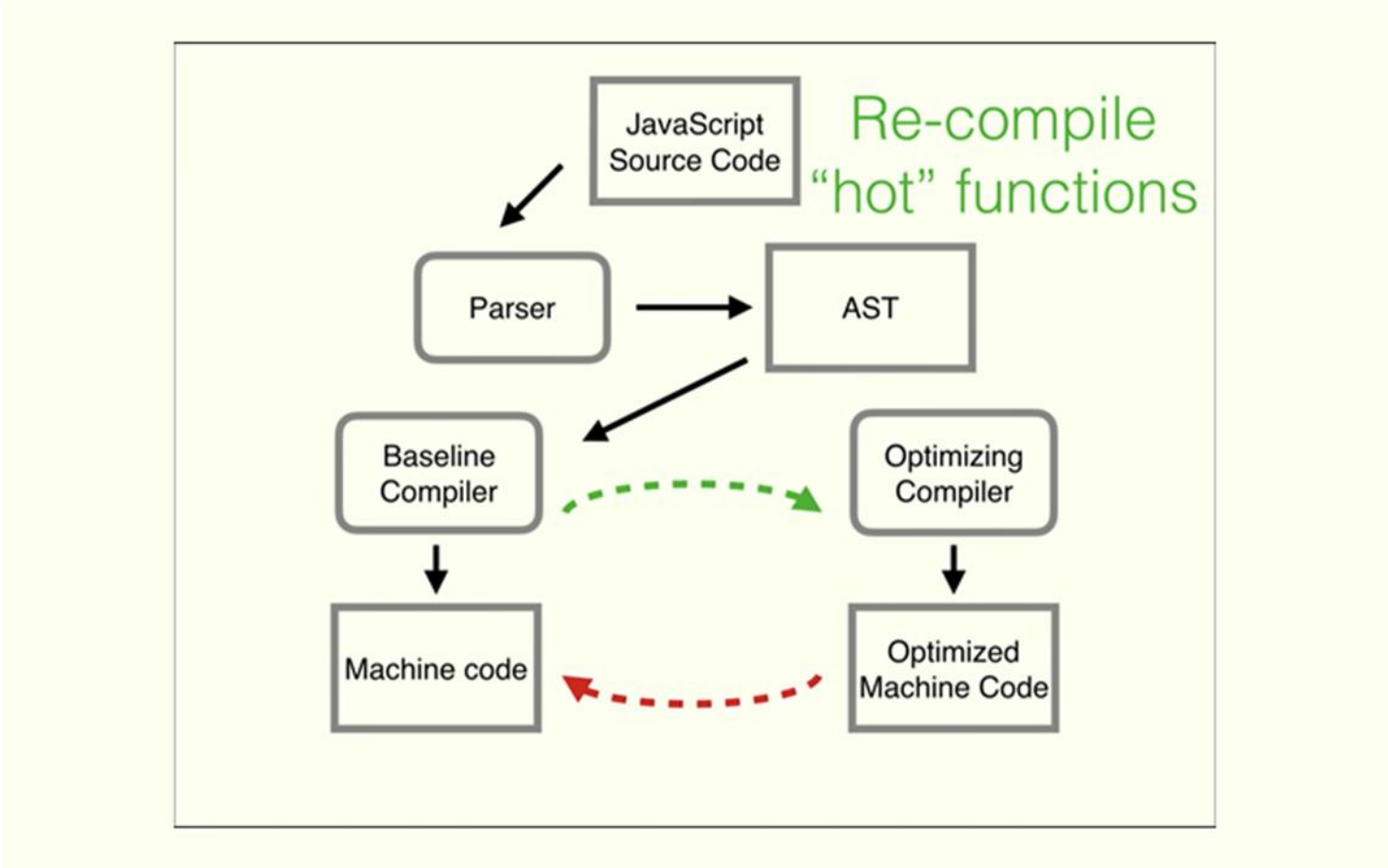
Portable compilation target
for high-level languages
like C / C++ / Rust

Open Standard

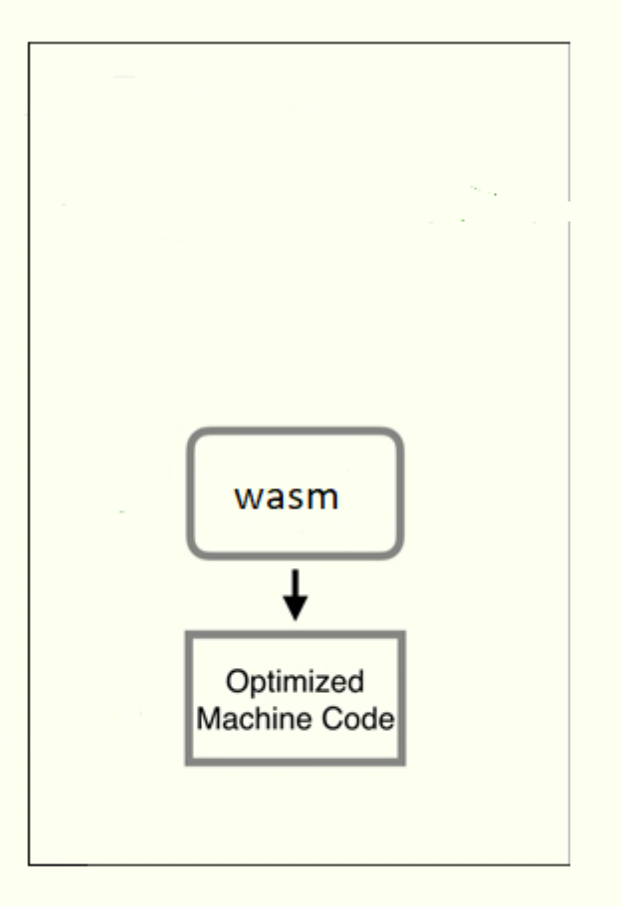
Why?
Performance
Safety

https://commons.wikimedia.org/wiki/File:Web_Assembly_Logo.svg

JavaScript execution

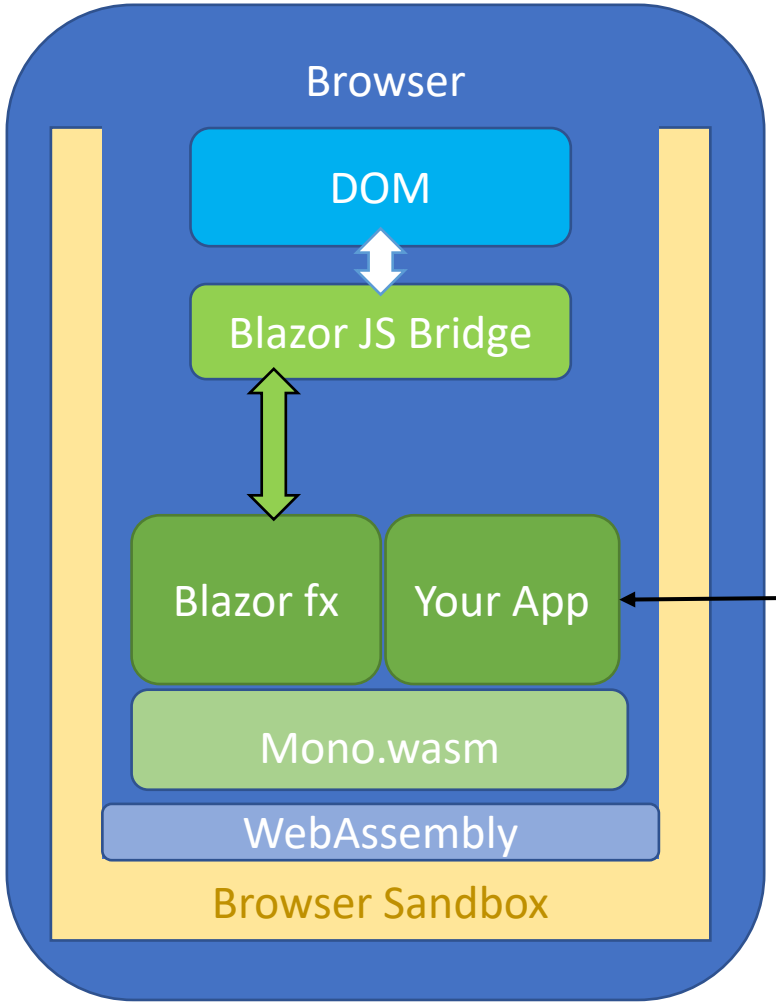


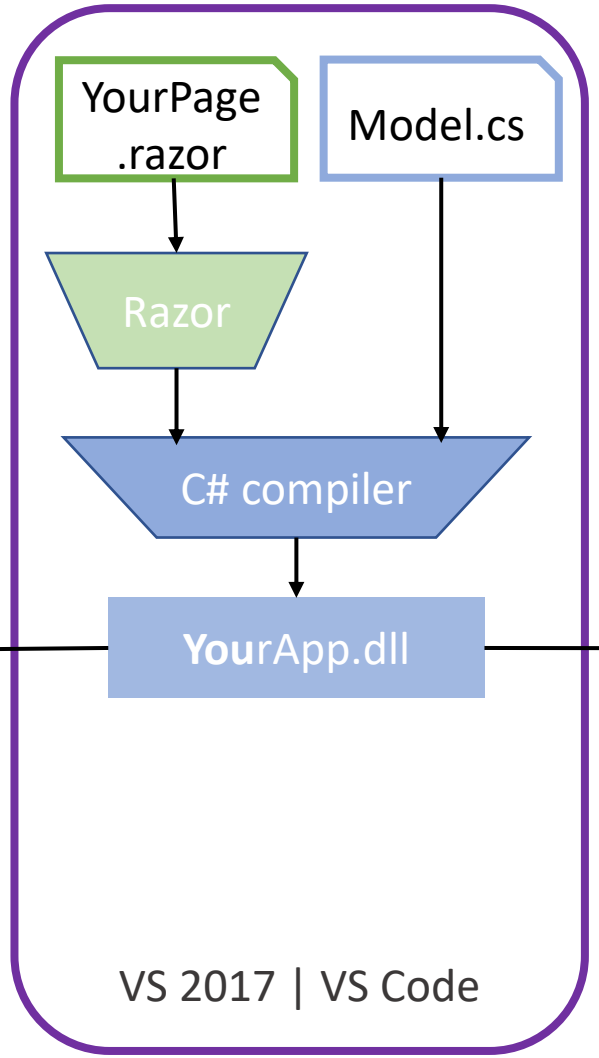
All the optimizations and recompilation happen in the user's time



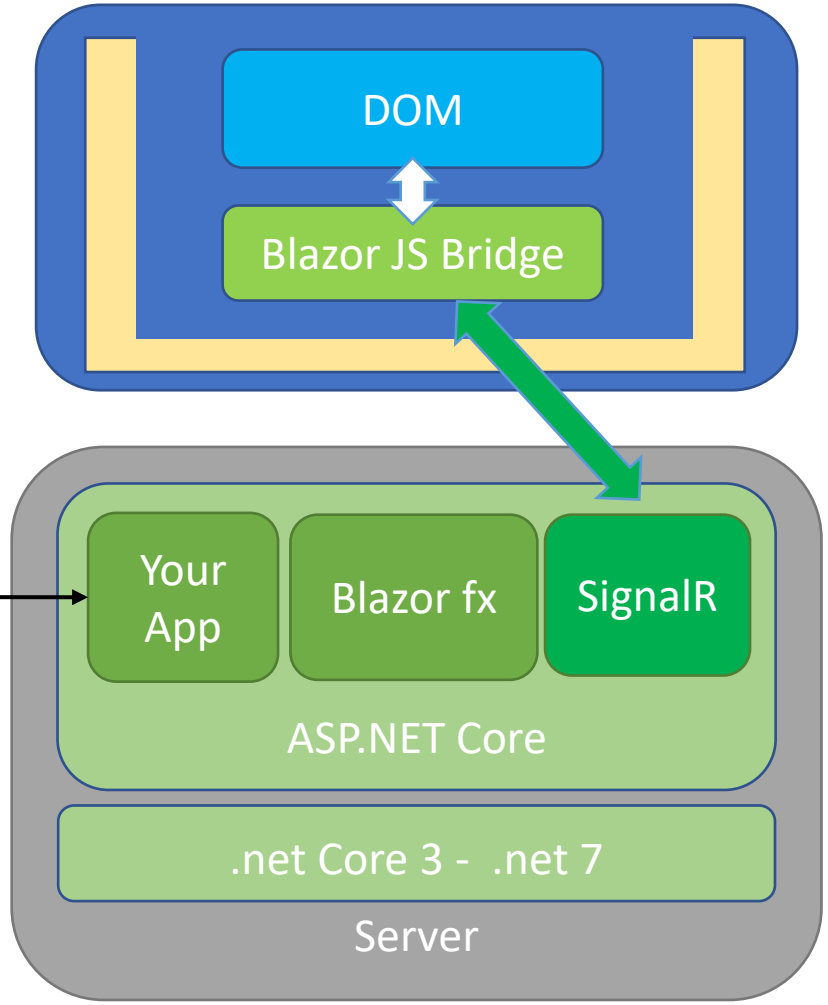
At deployment time

Blazor Client Side





Blazor Server Side



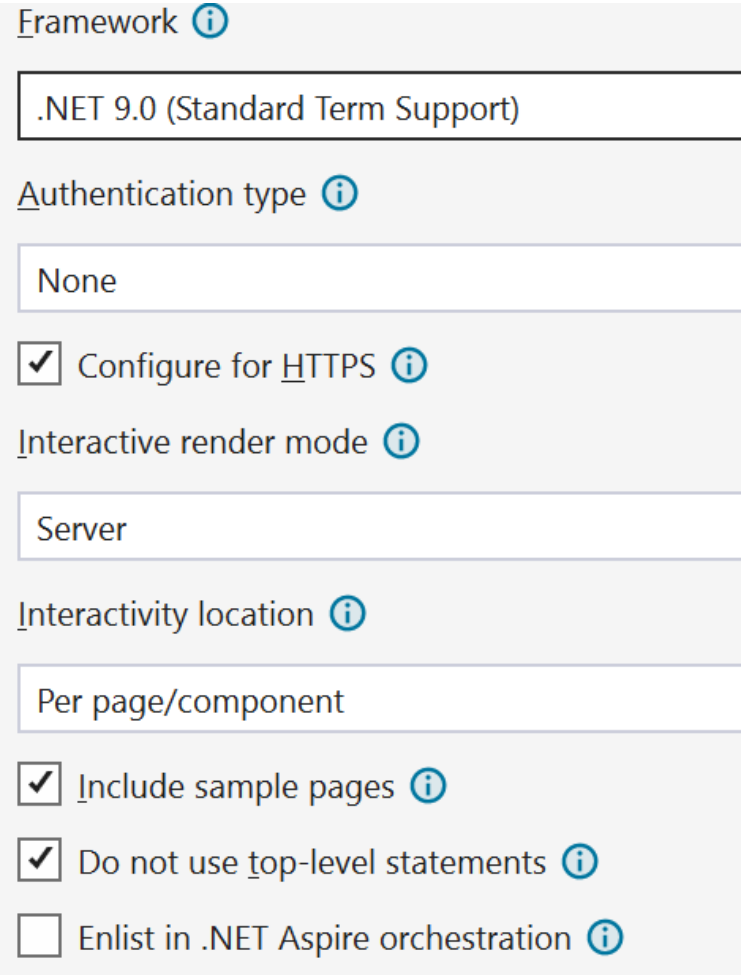
Blazor Rendermodes

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.

Hands on: Create a New Project

- Create a new “**Blazor Web App**”
- Select a folder for this Course, and add for example “\Day1”
- The name “BlazorApp1” is fine
- Use these settings ---------->



- Click [Create]
- Run and view the pages.

Investigating the sample app

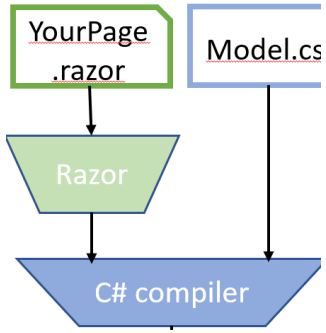
- Start in Program.cs, is the Register/Build/Use/Run pattern clear?
- Open `Components\App.razor`
- Run the App, right-click in the Browser and select **View Source**
- Open `Components\Routes.razor`, `Components\Layout\MainLayout.razor`, `Components\Layout\NavMenu.razor` and `Components\Pages\Home.razor`
- Do a quick check how they all end up in the source view. Don’t bother about the details yet. Look for `<main>` and `<article>`.

A few experiments

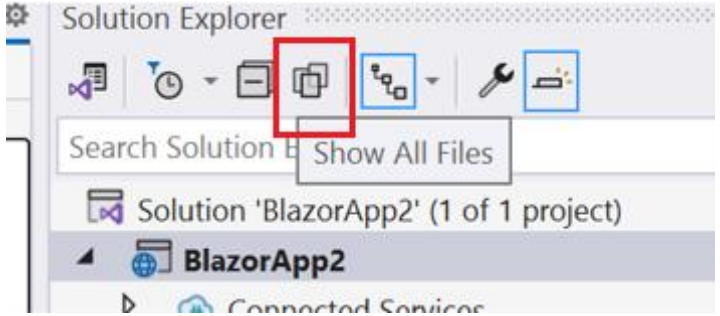
- Interactivity: study the **Counter.razor** page.
- A ‘Page’ is just a component with a **`@page "/"`** directive.
  - Open **Home.razor** and add **`<Counter />`** at the bottom. Run.
  - Use both Counters a few times, verify that they have no lasting memory.
- Open **Weather.razor**. Note that this is a **static rendering** page.
  - Find the Task.Delay() statement and Increase to 1500ms
  - Run, switch to the page a few times.
  - Remove the StreamRendering attibute, Run again.

- Add an **About.razor** page and add it to **NavMenu.razor** .

Razor and C# code combined

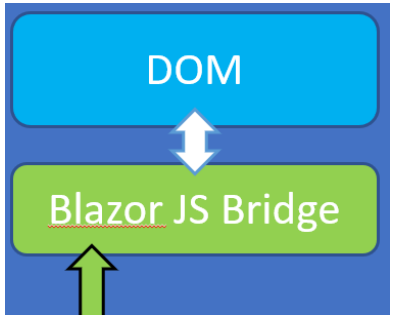


- In the current solution, open the project file (XML code)
- In the first Property group, add
  `<EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>`
  (intellisense wil help)
- Build the project and click 'Show all files'
- Type `.g.cs` in the Solution Explorer search box
- Open **`Components_Pages_Counter_razor.g.cs`**
- Compare this generated code with the source



About the RenderTree

- JavaScript operates on the **Document Object Model**, a tree of html objects.
- Blazor maintains its own copy, the virtual DOM.
- A ‘render’ means rebuilding (parts of) that virtual DOM.
- All detected changes are applied to the actual DOM, with the JS-Bridge



Code-behind files

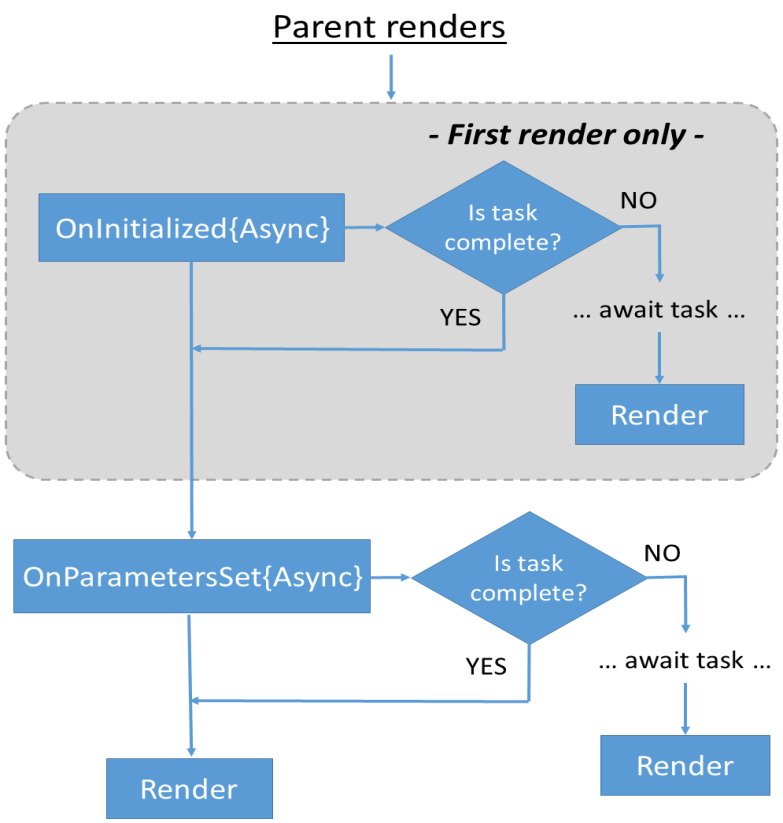
- Open **Counter.razor**
- Put your cursor inside the @code text.
- Type Ctrl+. and select “Extract block to code behind”
  Is the “ partial class ” feature familiar?
- Namespaces and Class names are derived from the folder structure.

- Note: in courseware and demos I try to avoid code-behind.
  In a real application it is the preferred way to work.

The Event Lifecycle, pt 1: Initialization

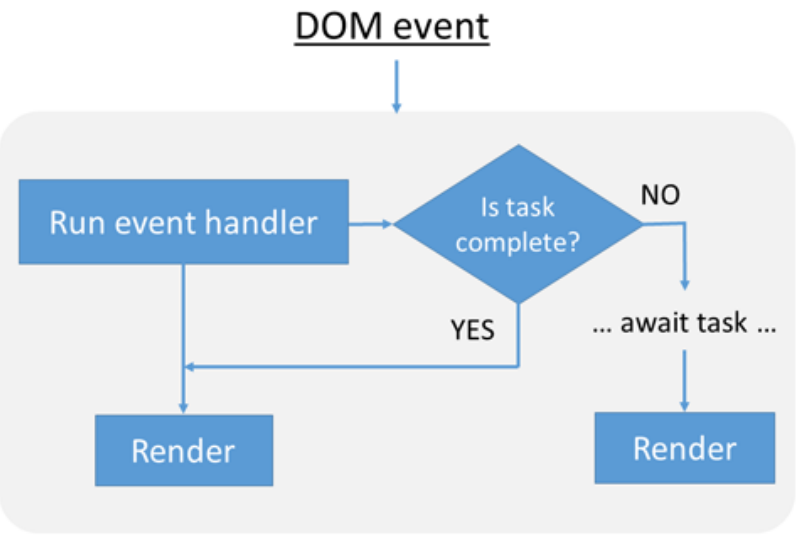
- OnInitialized (1 time) and OnParametersSet (1+ times) are the two main init events.
- They both have ...Async versions
- When you use an **`await`** then a Render may happen *before* OnInitializedAsync is finished.

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



The Event Lifecycle, pt 2: Events

- Blazor handles events with its own Eventcallback structure.
- An EventCallback accepts both `void` and `Task` returning methods.
- And again, during an `await` a Render can happen before the event is finished.



- Avoid `async void`. You won't need it.

The Event Lifecycle, pt 3: A practical example

- The goal: disabling a button when work is in progress.
- Create a new Project, with the same settings as before.
  Name: ButtonDemo, Server and Include sample pages.
- In Counter.razor, change:

```
private void IncrementCount()
{
   Thread.Sleep(1500);   // simulate CPU-bound work
    currentCount++;
}
```

- Run and check the ‘User Experience’.

The Event Lifecycle, pt 3: A practical example

- Make further changes:

```
<button disabled="@isBusy"  … >
…
bool isBusy = false;
private void IncrementCount()
{
   isBusy = true;
   Thread.Sleep(1500);       // simulate CPU-bound work
   currentCount++;
   isBusy = false;
}
```

- Run and check the ‘User Experience’ again. Not good yet.

The Event Lifecycle, pt 3: A practical example

- Make further changes:

```
private async Task IncrementCount()
{
   isBusy = true;

   StateHasChanged();   // isBusy has changed
   await Task.Yield();  // 'invite' a render to update the button

   Thread.Sleep(1500);  // the delay
   currentCount++;

   isBusy = false;
}
```

- Check the UI
- Improve with a try/finally block and an if(isBusy) return; guard.
- Use View Source in the Browser to see what Blazor does with **disabled="true** or **false"**

A simple component: A Clock

- Create a new Project, name it ClockDemo
- Add a Controls folder under Components
- Add a razor component under Controls , **Clock.razor**
- Replace the <h3> with

```
<div>
    It is: @DateTime.Now.ToLongTimeString()
</div>
```

- Add **<Clock />** to the Home page, open and fix _Imports.razor
- Run, switch pages a few times

Making a Clock, pt 2

- Add the following:

```
@code {
    Timer timer = default!;
    protected override void OnInitialized()
    {
        timer = new(OnTick, null, 0, 1_000);
    }
    private void OnTick(object? state)
    {
        StateHasChanged();
    }
}
```

- Run. Expect an exception.

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)

Exercise: make a CountDown Control

- Make a component with a button and a text field (remaining time)
  - Hint: @($“Time left: {remain:0.00}")
- A click should show a countdown from 10 seconds to zero.
- Do this **without a Timer**.
- Use a loop and Task.Delay(500) instead.
  Use DateTime.Now to calculate the remainder.

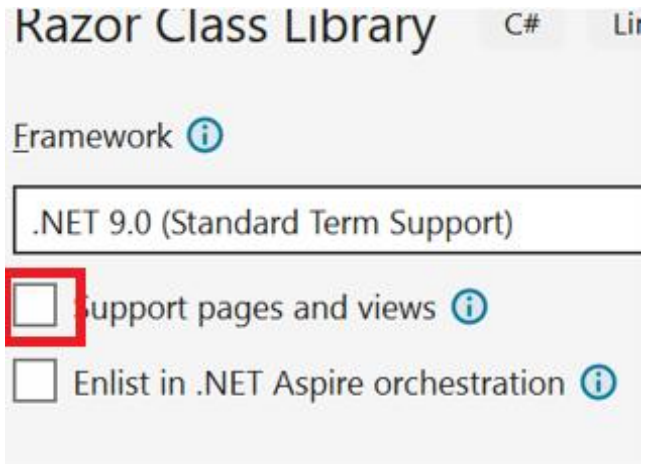
- Add and display a loop counter.
  Then reduce the 500 in steps, see how well Blazor (SignalR) keeps up.
  Finally, replace Delay(1) with Task.Yield()

Agenda

- Depending on the time we do:
- Add controls to a Razor Class libary (short)
- Create a composite control (TabPages) (longer)

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
- Study Component1.razor and the linked css



Using a Razor Class Library pt 2

- Add **<MyComponents.Component1 />** to Home.razor
  You can add the RCL namespace to _Imports.
- Run and see that this works “out of the box”
- Add the Clock or Coutdown component to the RCL.
- Use it on the Home page.

- Strange Use-case: Blazor will alwyas re-compile all razor files. When you have a project with hundreds of pages compiling becomes very slow. Splitting it into RCLs can solve this.

Making a TabPages control

We look at how Components work together. Same as Grid&Columns.

- Create a new Project, TabsDemo
- Add a Controls folder
- Add TabControl.razor and TabPage.razor. Leave as-is.
- Use it in Home.razor:

```
<TabControl>
    <TabPage> Inside </TabPage>
</TabControl>
<TabPage> Outside </TabPage>
```

- Try to run, there is an exception

A TabPages control, continued

- The first error is that TabControl does not have a ChildContent parameter. Let's add it, inside @code:

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

- Run again. Looks the same but now it's about the TabPage.
- Add the same code to the TabPage. Run again.
- No errors, no "Inside" or "Outside" text.

A TabPages control, Parent/Child relations

- Blazor allows you to ‘flow’ a parameter down the RenderTree.
- We use this to make the TabControl pass itself. Replace the <h3> with:

```
<CascadingValue Value="this" IsFixed="true">
    @ChildContent
</CascadingValue>
```

- IsFixed is an optimization we tell Blazor not to watch for changes.
- Note that the output now shows all TabPages.

A TabPages control, Parent/Child relations

- In the TabPage control, add:

```
<div>@ChildContent</div>
```

- and

```
[CascadingParameter]
public required TabControl Parent { get; set; }

protected override void OnInitialized()
{
   // ArgumentNullException.ThrowIfNull(Parent);
}
```

- Run this first without and then with the ArgumentNullException.
- The error is correct, (re)move the “Outside” TabPage.

A TabPages control, connecting

- Change the TabPage:

```
[Parameter, EditorRequired]
public string Title { get; set; }

protected override void OnInitialized()
{
  ArgumentNullException.ThrowIfNull(Parent);
  Parent.AddPage(this);
}
```

- Add AddPage() and a private List<TabPage> to the TabControl. Simple C#, no code provided.
- In Home.razor, add Titles to the pages.

TabControl, first attempt to show the Titles

- We will use a simple approach first. In TabControl.razor

```
<CascadingValue Value="this" IsFixed="true">
    <ul>
        @foreach(var page in pages)
        {
            <li>@@page.Title</li>
        }
    </ul>
</CascadingValue>
```

- @page is a reserved word, escaped with @@
- Run. Do we see anything?

TabControl, what went wrong?

- The current TabControl does not render its @ChildContent
  - And therefore the TabPages are never created. Check with a breakpoint.
- Second attempt:

In TabControl

```
<ul>
   @ChildContent
</ul>
```

In TabPage

```
@* <div>@ChildContent</div> *@
<li>@Title</li>
```

- Now you should see the Titles, as a bulleted list.
- The list of pages is no longer required, remove it. And the AddPage().

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);
```

- Add IsActivePage and SetActivePage to the Parent control.

TabControls, the End

- Add some more pages to Home.razor:

```
<TabControl>
    <TabPage Title="Insider"> Inside </TabPage>
    <TabPage Title="Outsider"> Was Outside </TabPage>
    <TabPage Title="Counter"><Counter /></TabPage>
    <TabPage Title="Weather"><Weather /></TabPage>
</TabControl>
```

- A RenderFragment can be any markup, from “” to multiple elements.
- Put a breakpoint in Weather.OnInitializedAsync, check when it runs.