

# Blazor

Dag 1 van 3

# Agenda

## Dag 1

- Wat is Blazor?
- Hoe werkt Blazor ongeveer?
- Event LifeCycle
- Componenten maken
- Razor Class Library

## Microsoft Web technologies

- 1997 .asp Active Server Pages
- 2001 .aspx WebForms
- 2007-2012 .xaml SilverLight
- 2008 .cshtml MVC, Model View Controller
- 2012 .cshtml Razor Pages
- 2017 .razor Blazor (aka Razor Components)
- 2023 Q4 .razor Blazor United

**WebForms** was het vlaggeschip van .NET 1, vooral goed in afstand nemen van HTML (HTML4 kwam in 4 heel verschillende standaarden).

**MVC** is veel sneller (lichter) , in feite een Web API die HTML levert. *Stateless*.


**Razor Pages** is een soort “MVC Light”, minder code.

**Blazor** is een SPA (net als Angular, React etc) maar met C# en weinig/geen JS

**Blazor Server** is een pseudo-SPA, vrij uniek. Wel zwaar voor de Server (State en SignalR circuit voor iedere user)

**Blazor United** (dotnet 8) in 1 App een combinatie van Blazor Client-side, Blazor Server-side en (een soort) Razor Pages vloeiend verbinden. [Blazor United prototype - YouTube](#)

**Razor** is de CSHTML compiler (pre-processor) die naast MVC en Razor Pages ook in Blazor gebruikt word. De file extensies zijn een soort historisch ongelukje.



## 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

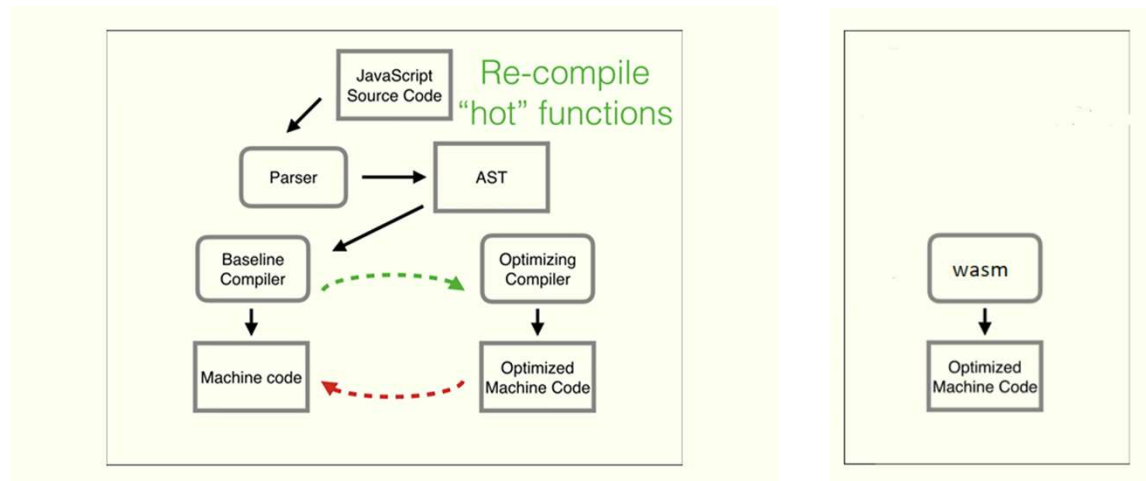
Why?

- Performance
- Safety

[https://commons.wikimedia.org/wiki/File:Web\\_Assembly\\_Logo.svg](https://commons.wikimedia.org/wiki/File:Web_Assembly_Logo.svg)

Er zijn diverse Tool/Compiler kits beschikbaar voor unmanaged talen, ook voor C

# JavaScript execution



Uitvoering van JavaScript is een belangrijke feature van iedere Browser.

De optimizing compiler en de instrumentatie kosten ook tijd.

Bij gebruik Wasm wordt alle optimalisatie vooraf gedaan, voor eindgebruiker: download and run.

## AST Abstract Syntax Tree (Compiler datamodel)

# Some Facts about WASM

## Very different from .NET's IL

- Much simpler
- Linear memory
- No GC

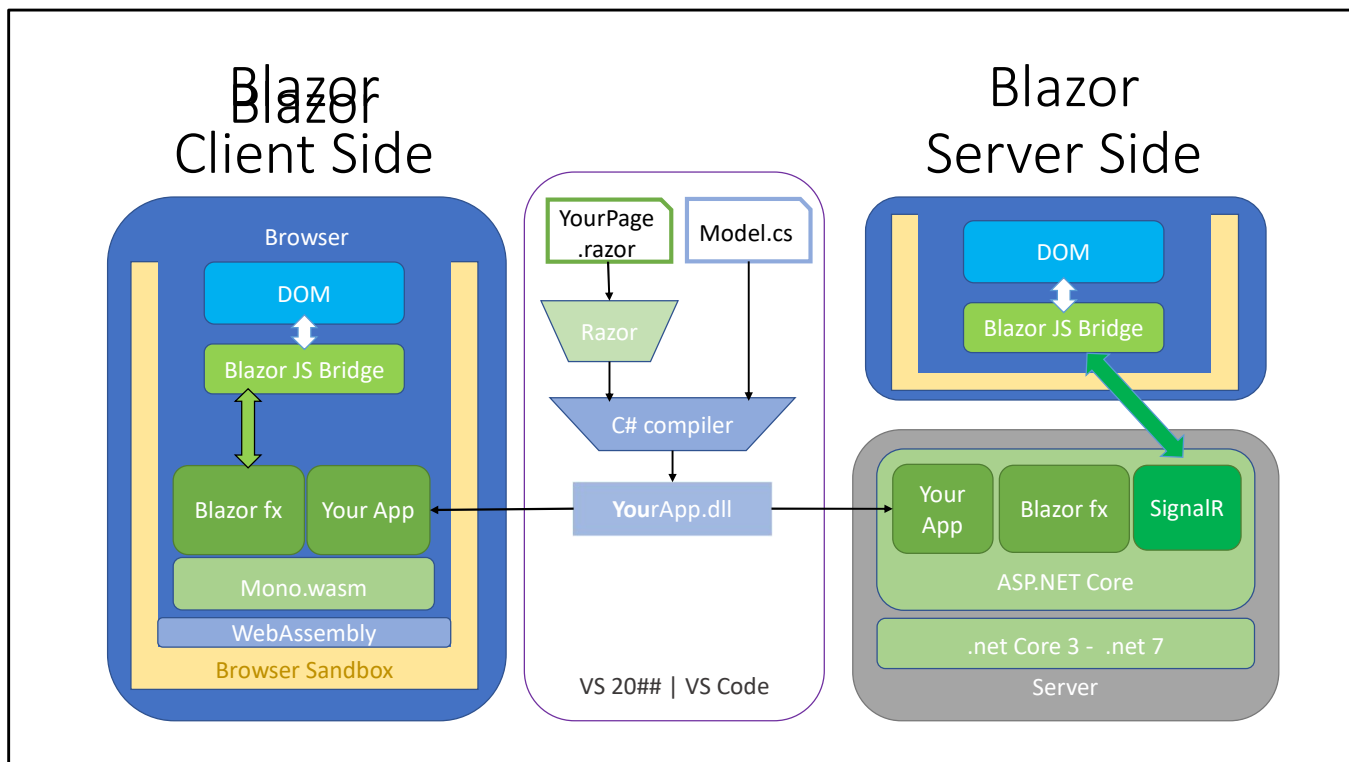
## Cannot access the DOM = no UI

- (...yet)

## JavaScript interop exists

- WASM calls JS
- JS calls into WASM

Unreal en Unity-3D draaien inmiddels op Wasm.



Blazor is bedacht door Steve Sanderson

**Blazor Server** was in eerste instantie een “developer feature” maar er bleek een markt voor te zijn.

**Mono** was de “open source dotnet” implementatie. De basis voor Xamarin. Sinds .net 5 vervangen door de “dotnet portable runtime”

**DOM:** Document Object Model (HTML als objecten).

**JS Bridge:** WebAssembly code mag **niet** rechtstreeks bij de DOM (UI en I/O)! Dit is een security feature. Het moet via JS interop.

**SignalR** is een snel two-way communicatie protocol op basis van WebSockets

De **Browser Sandbox** is het JavaScript security cordon. Merk op dat YourApp.DLL in Blazor Server buiten dat cordon draait.

**fx** is Amerikaans voor framework

## Client-side of Server-side

Onderwerp	Client (WebAssembly)	Server (.net)
Startup	Traag (vooral 1e keer)	Snel
Browsers	Alle grote recente	Ook oudere (IE), alleen JS nodig
Offline bruikbaar	Ja, zelfs PWA	Nee
Vriendelijk voor SEO (Google)	Beperkt, met "Server pre-rendered"	Ja, met pre-rendering
Geheimhouden source code	Nee	Ja
Schaalbaarheid	Giga	Kilo
Direct naar Database	Alleen lokaal (Sqlite, BrowserDb)	Ja
Andere (API) services	Alleen HTTP en CORS Policy	Onbeperkt
Voor programmeur	Extra API tussenlaag nodig	Compacter, 3 lagen in 1 project

Alles wat je in de Browser laadt "ligt op straat".

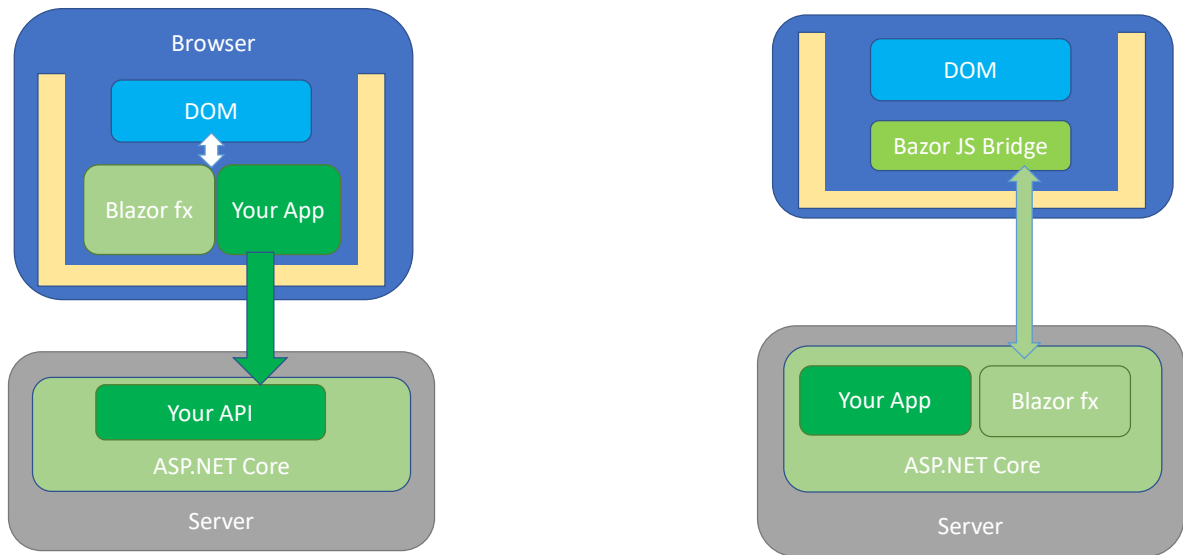
Blazor Wasm met bijv MS-SQL : a) wachtwoord niet geheim en b) protocol niet toegestaan

Maar:

- De meeste voordelen van Blazor Server heb je ook als je Blazor Wasm combineert met een API server.
- Wat overblijft is minder complexiteit en overhead



## De eigenlijke keuze

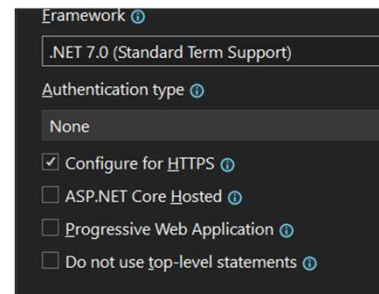


Een Blazor Wasm App zal vrijwel altijd gecombineerd worden met een API server.

Daarmee heb je de meeste voordelen van Blazor Server ook daar. Maar het is wel meer werk.

## Mee-doe oefening: Hoe start Blazor

- Maak een Blazor WebAssembly project
- Kies een naam en een Folder
- Doe de stappen mee of kijk op scherm
- We bekijken
  - index.html
  - App.razor
  - Program.cs



Een oefening / Demo

Je kunt meedoen of achterover leunen en alleen kijken

Optioneel: RootComponent, PageTitle, HeadContent

## De RenderTree (virtual DOM)

- Blazor genereert een eigen DOM uit de Razor code
- Bij het opbouwen van een nieuwe DOM wordt die vergeleken met de bestaande
- De verschillen worden met JS Interop in de echte DOM aangebracht
- Demo: we bekijken de gegenereerde C# code

In de .csproj file:

```
<PropertyGroup>  
  <EmitCompilerGeneratedFiles>true</EmitCompilerGeneratedFiles>  
  <CompilerGeneratedFilesOutputPath>.</CompilerGeneratedFilesOutputPath>  
</PropertyGroup>
```

Het default OutputPath is `<project>\obj\debug\net7.0\generated\...`

Code generators gaan voor betrouwbare lange namen.

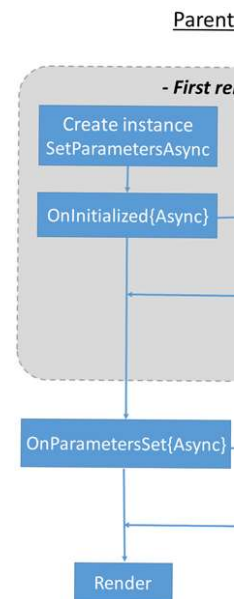
Vergelijk **Counter.razor** met **Pages\_Counter\_razor.g.cs**

Optie: Code-behind class maken

De RenderTree is geen “Shadow-DOM”, dat is een JS feature dat gaat over CSS isolatie.

## De Event LifeCycle zonder Async

- De events komen in Sync en Async varianten
- Alles Sync : simpele rechte lijn
  - OnInitialized: eenmalig
  - OnParametersSet: bij een update  
Er is een parameter gewijzigd
  - OnAfterRender: na een Render  
Component geladen, JS beschikbaar



De non-async versies tonen de basis volgorde.

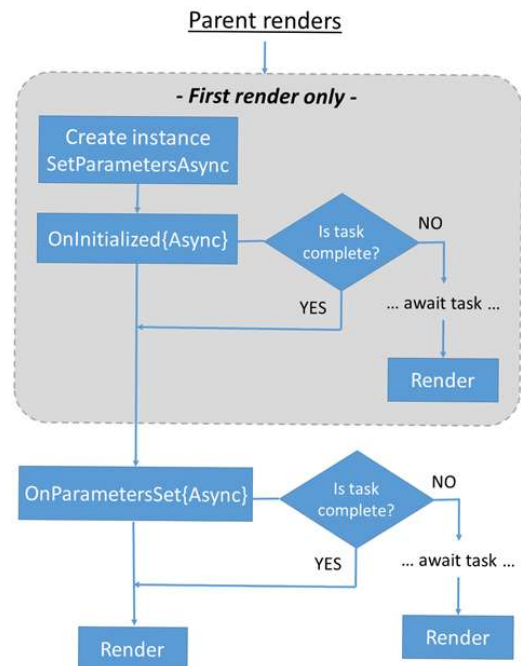
**SetParametersAsync** zul je vrijwel nooit nodig hebben.

Als je het wel gebruikt, lees dan eerst de docs heel erg goed. Er zitten ook tijd-gevoelige (!) aspecten aan.

# De volledige Event LifeCycle

- Er kan al een Render plaatsvinden tijdens OnInitializedAsync of OnParametersSetAsync
- Niet iedere **async Task** method is ook echt asynchroon

Bron: [Lifecycle events](#)



De andere events (@buttonclick, @onchange etc) behandelen we later, samen met EventCallback.

## Experiment met FetchData

- We bekijken FetchData in de standaard Blazor Server App
- Waarom is de null-check nodig? Is deze wel nodig?

Dit is geen Blazor Wasm / Blazor Server issue.  
Blazor Server gebruikt een

**`Task<WeatherForecast[]> GetForecastAsync(DateOnly startDate)`**  
die niet echt asynchroon is. In de andere templates is de null-check wel nodig.

Optie: verder uitdiepen async vs asynchroon, google evt ook “eliding async/await”

# Oefening met Counter.razor

## De opdracht

- Start een nieuw project en pas de Counter page aan zodat:
- Na 1 klik gaat de counter 5x omhoog met een seconde tussentijd

## Hints

- `IncrementCount()` mag je async maken, de razor compiler past zich aan. De `<button>` niet wijzigen.
- Gebruik geen `Timer`, `Task.Delay(1_000)` wacht 1 seconde.

Conceptueel krijg je 2 `StateChanged()` calls cadeau: 1 voor en 1 na het event.

Voor de bonus punten: disable de button gedurende die 5 seconden

Suggestie: **`private bool busy = false;`**

en `<button>` nu wel een beetje aanpassen: **`disabled="@busy"`** (bekijk dat ook in de Dev Tools)

Probeer ook: 1 Klik is 1000 stappen, maak het nu zo snel mogelijk maar wel zichtbaar oplopend.

Meet dan ook de tijd voor 1000 stappen, **`var watch = System.Diagnostics.Stopwatch.StartNew();`**

# Componenten

- Terzijde: een Page is een Component met 1 of meer `@page “/...”`
- File naam = class name moet beginnen met een Hoofdletter
- Korte Demo: gebruik van Counter als Component
- Wat niet mag (compileert wel werkt, niet) **`var counter = new Counter();`**  
Componenten worden altijd aangemaakt in BuildRendertree



## Oefening: Een Real-time clock

We maken een herbruikbare Clock component

- Start een nieuw WebAssembly project.
- Voeg een Razor Component toe, **Clock.razor** .
- Add `private System.Timers.Timer? timer;`  
en initialiseer die in `protected override void OnInitialized();`
- Implementeer het Elapsed event (Intellisense haakt hier af)
- Implementeer IDisposable voor de timer.  
Begin met `@implements IDisposable` bovenin.
- Maak een simpele UI en test de Clock op Index en Counter

De UI is hier niet het doel.

- Timer.Elapsed is **geen** Blazor (LifeCycle) event.
- Daarom zelf StateHasChanged() uitvoeren

Vervolgstap:

- Maak een Blazor Server project (mag in zelfde Solution).
- Copy/Paste de Clock component en test deze (ik verwacht nu een exception).

## Razor Class Library

- Demo om mee te doen

## Parameters en EventCallback

- Demo

## De Blazor Wasm Hosted Template

- Een vrij standaard patroon voor gebruik Blazor Wasm
- We bekijken de 3 Projecten en de samenhang
- We voegen een Controller toe

## Dag 2 / uitloop

- Maak een Razor Class Library (RCL) voor de Clock  
Bekijk hoe de resources worden doorgegeven
- Parameters en EventCallback (Demo)
- Voorbespreken (keuze onderwerpen)
  - TabPages (CascadingValue)
  - Een DIY Modal Dialog (zonder service)
  - Een Table component (simple Grid)  
Daarna deep-dive QuickGrid en Virtualize

Het is een Agile cursus. Wat zijn de User Stories?

## Een Tab Control

Dit demonstreert o.a.

- het gebruik van CascadingValue
- Component interactie in C# (Parent / Child pattern)
- RenderFragments

Het Parent / Child pattern is ook de basis voor bijv DataGrid ColumnDefinitons  
CascadingValue is ook een hoeksteen van EditForm

## De volgende dagen

- Data binding, EventCallback
- <form> en <EditForm>
- Testen
- State management
- DI Scopes (DbContext, IDisposable) (Blazor Server)
  
- Niet om gevraagd (?)
  - Authentication
  - Architectuur

## Github

- <https://github.com/henk787/Blazor-DSE.git>