# WebAssembly & Blazor

## Ready for Enterprise?

#shapingthefuture


ANGULARJS
by Google

# About me & MAXIMAGO

## Maik Schöneich

- Born 1983
- Finished education in 2006
- .NET Developer since then
- Web-Frontend developer since 2013

## MAXIMAGO

- UI Special Unit
- Based in Lünen (near Dortmund)
- User centered software development
- Business Software as simple as Apps


WARNING
This is a highly opionated talk.
Discussion is appreciated
afterwards.
OSI 3598


# WebAssembly

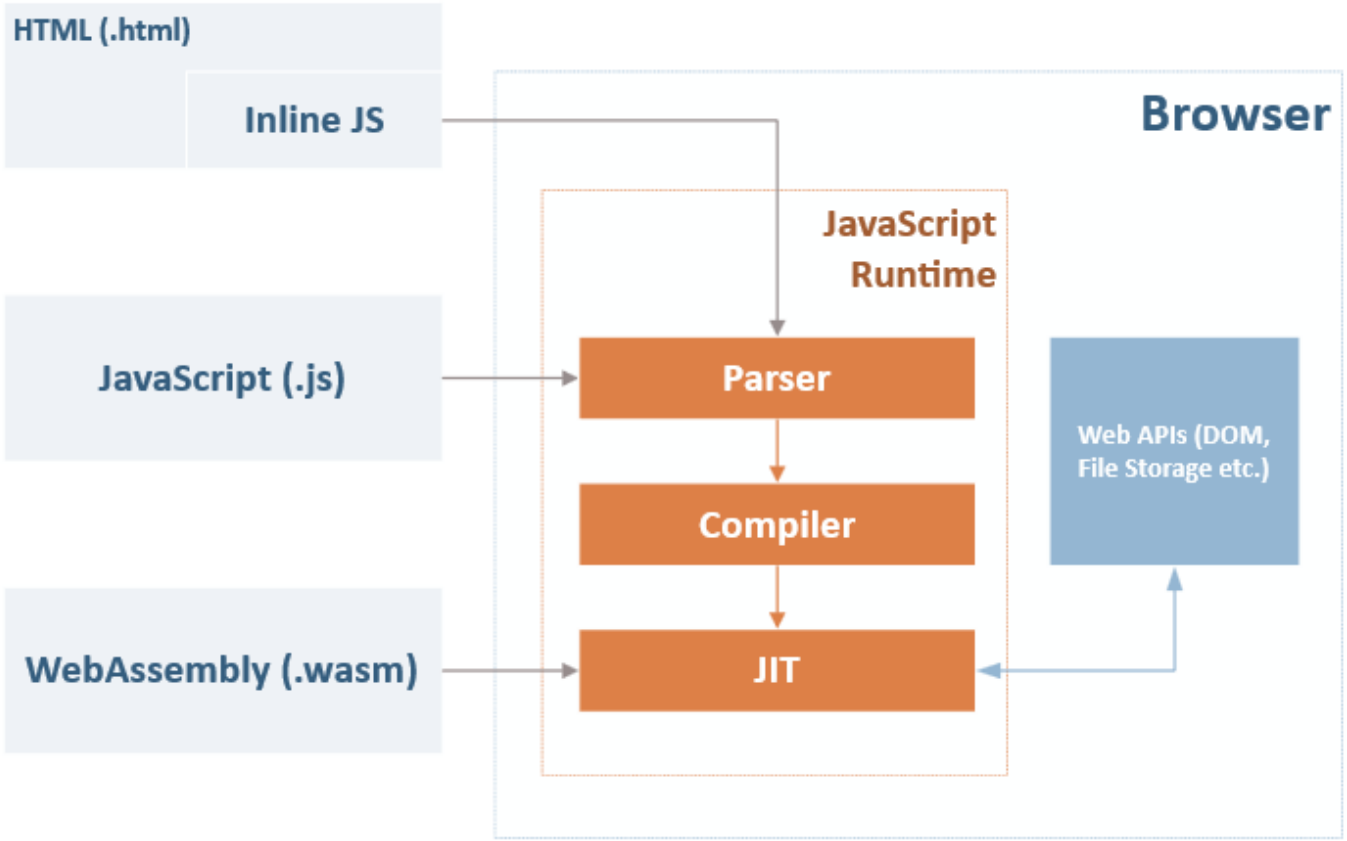
Native code in the browser

# What is WebAssembly?

"WebAssembly is a binary instruction format for a stack-based virtual machine."

https://webassembly.org/

# WebAssembly




Source: https://daveaglick.com/posts/blazor-razor-webassembly-and-mono

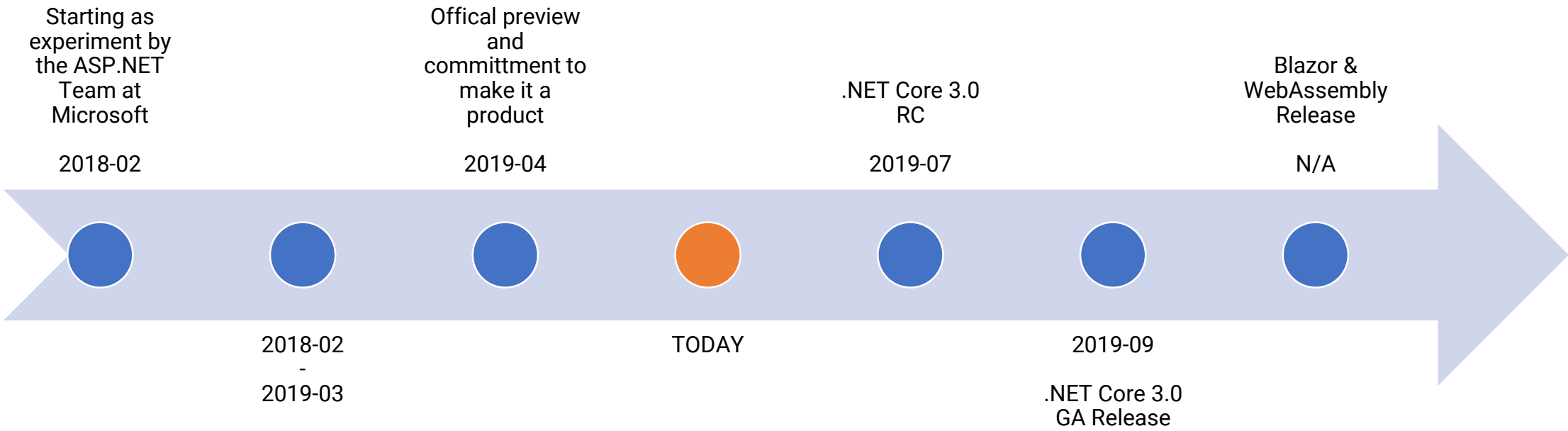
# Blazor

Client-side running with WebAssembly in the browser

# What is Blazor?

- Interactive WebUI with C#
- Run in WebAssembly or on the server
- Built on open web standards – No plugins needed!!!
- Use .NET libraries based on .NET Standard 2.0
- JavaScript Interop

# Timeline


Starting as experiment by the ASP.NET Team at Microsoft
2018-02
2018-02 - 2019-03
Versions 0.2.0 to 0.9.0 continously enriched and developed
Offical preview and committment to make it a product
2019-04
TODAY
.NET Core 3.0 RC
2019-07
2019-09
.NET Core 3.0 GA Release
Blazor & WebAssembly Release
N/A

# Getting started

1. Install latest .NET Core 3.0 Preview

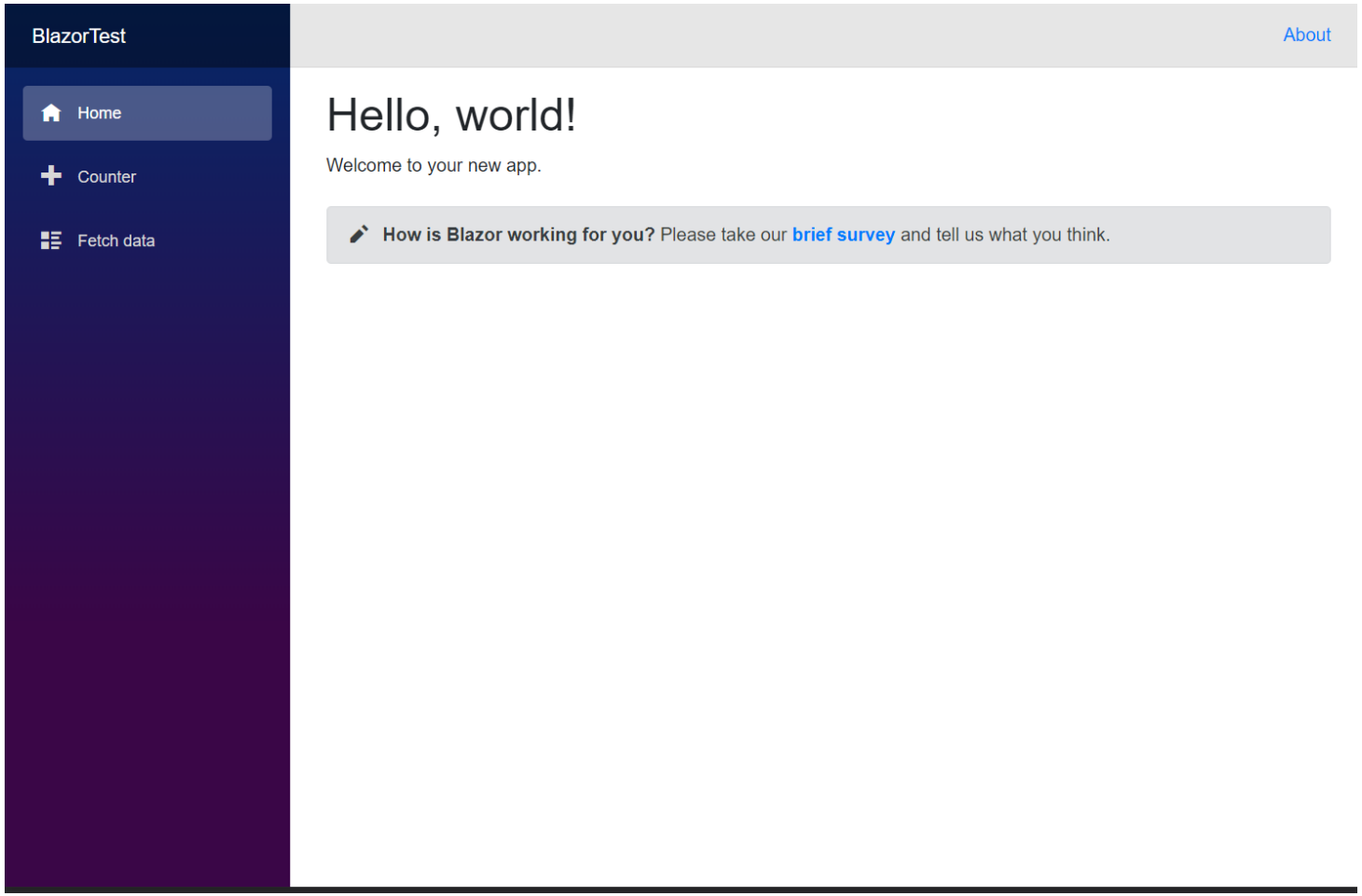
2. Install Blazor Templates

```
dotnet new -i Microsoft.AspNetCore.Blazor.Templates::3.0.0-preview6.19307.2
```

3. Create a first app using Visual Studio 2019 Preview

https://docs.microsoft.com/de-de/aspnet/core/blazor/get-started

# First application



# Components

```
<!-- Sample HTML Content of a page -->

<h1>Show the use of Components</h1>

<!-- Use a blazor component -->
<blazor-panel Title="how to use blazor components“
              OnClick="@Execute">
    <span>
        Some other content
    </span>
</blazor-panel>

@code {
    private void Execute()
    {
        // Do some stuff
    }
}
```

# Components

```
<div class="panel panel-default">
    <div class="panel-heading">@Title</div>
    <div class="panel-body">@ChildContent</div>
    <button class="btn btn-primary" @onclick="@OnClick">
        Trigger a Parent component method
    </button>
</div>

@code {
    [Parameter]
    private string Title { get; set; }

    [Parameter]
    private RenderFragment ChildContent { get; set; }

    [Parameter]
    private EventCallback<UIMouseEventArgs> OnClick { get; set; }
}
```

# „Code-Behind“ Style

```
<!-- MyComponent.razor -->

@inherits MyComponentBase

<div class="panel panel-default“>
    <p>@MyContent</p>
</div>



// MyComponentBase.cs

public class MyComponentBase : ComponentBase
{
    public string MyContent { get; set; } = "Blazor rocks the browser!";
}
```

# Data Binding & Event Handling

```
<!-- Data Binding -->

<input type="text" id="text" @bind="SampleText" />

<input @bind="TalkDate" @bind:format="yyyy-MM-dd" />

<MyComponent @bind-ShowComponent="IsActive" />

@code {
  public string SampleText {get;set;} = "My Text";

  public DateTime TalkDate {get;set;} = new DateTime(2019, 6, 25);

  public bool IsActive {get;set;} = true;
}

// MyComponent.razor
@code {
  [Parameter]
  public bool ShowComponent {get;set;}
}
```

# Data Binding & Event Handling

```
<!-- Event Handling -->

<button class="btn btn-primary" @onclick="@Execute">
  Start
</button>

<input type="checkbox" @onchange="@CheckboxChanged" />

<button class="btn btn-primary" @onclick="@OnClick">
  Trigger a Parent component method
</button>

@code {
  private void Execute(UIMouseEventArgs e) { /* …Do something… */ }

  private void CheckboxChanged() { /* React to the change */ }

  [Parameter]
  private EventCallback<UIMouseEventArgs> OnClick { get; set; }
}
```

# Layouts

```
<!- MyLayout.razor -->
@inherits LayoutComponentBase

<div class="container">
  <nav>
    <a href="home">Home</a>
    <a href="other">Other</a>
  </nav>
  <div class="content">

    @Body

  </div>
</div>


<!-- Home.razor -->
@layout MyLayout
@page "/home"

<h1>Welcome to this page!</h1>
```

Layouts can be nested

# Dependency Injection

```
// Startup.cs

public void ConfigureServices(IServiceCollection services)
{
  services.AddSingleton<IMyService, MyService>();
  services.AddTransient<IMySecondService, SecondService>();
  services.AddScoped<IScopedService, ScopedService>();
}
```

Default services available:

- HttpClient
- IJSRuntime
- IUrlHelper

```
<!-- Home.razor -->
@page “/home“
@inject HttpClient httpClient

<h1>Welcome!</h1>

@code {
  private async Task DoSomething()
  {
    var response = await httpClient
     .GetAsync(“http://www.google.com“);
  }
}
```

```
// MyService.cs

public class MyService
{
  public MyService(HttpClient httpClient)
  {
    // Use constructor injection. Optional parameters
    // with default values don‘t need to be provided
    // by DI
  }
}
```

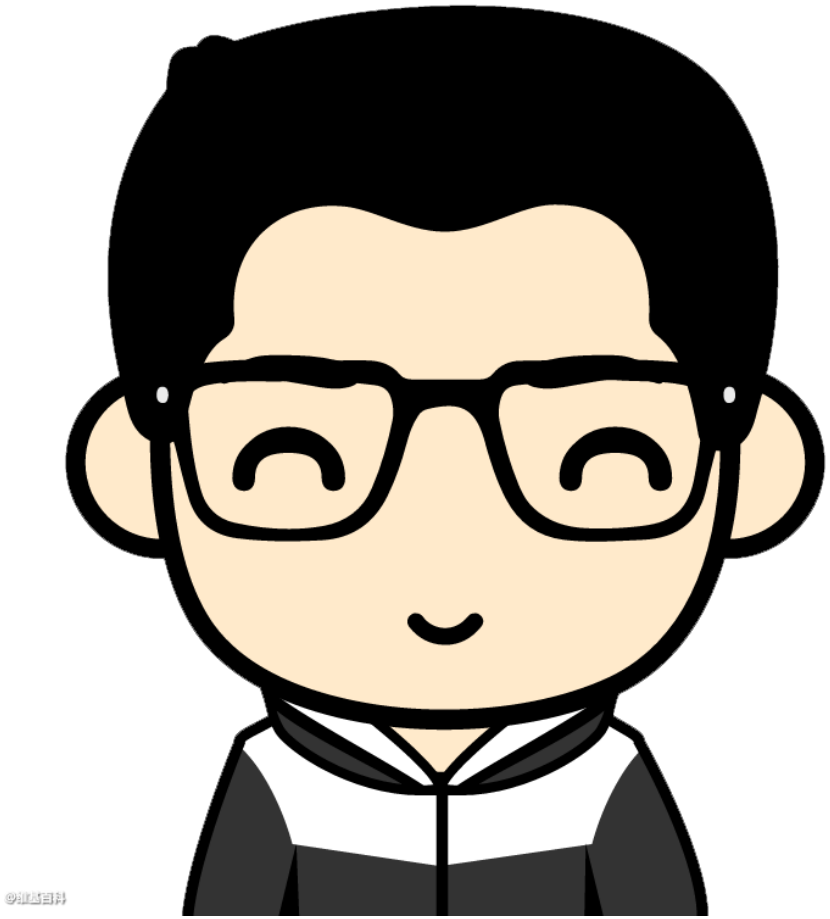
# Routing

```
<!-- App.razor -->
<Router AppAssembly="typeof(Program).Assembly" />


<!-- Error.razor -->
@page "/Error"
@page "/Error/{Message:string}"

<div class="error">
  <h2>Error</h2>
  @if(!string.IsNullOrEmpty(Message))
  {
    <div class="error-message">@Message</div>
  }
</div>

<NavLink href="Home">
  Back to Home
</NavLink>

@code {
  [Parameter]
  public string Message { get; set; }
}
```


# Blazor & Business Apps

## Ready for the Enterprise?

# Authentication & Authorization



# Authentication & Authorization

- Added in the latest Preview
- Mainly for Server-Side-Blazor
- Known mechanisms from ASP.NET
- Rule & Policy based authentication

# Demo

# Modularization

# Modularization

- Dividing complex systems
- Feature modules
- Lazy Loading
- Own internal structure

# Demo

# Visual Design

# Visual Design

- Impress with own design
- Comply to corporate design guidelines
- Show individual UI controls
  - Complex controls
  - Charts
  - DataGrids

# Demo

# Deployment

- Ship it as it is to any webserver you like
  - NGinX
  - Apache
  - Node http-server
  - IIS
- Maybe some configurations have to applied to support correct mime-types

# Demo

# Conclusion

# Well done

- Use C# and .NET ecosystem to build SPA
- No need to learn new language
- Reuse patterns and tools from existing platforms
  - Authentication & Authorization
  - Dependency Injection
  - Razor Engine & Syntax

# Things to consider

- Use Razor as template engine?
- Inheritance over composition?
- Generated code?

# Give it a try.

It has a big potential!

# Thanks for your time!

http://tiny.cc/dwx2019-blazor