The modern .NET

Real time web apps with Blazor and SignalR

Marian Filipovici

DIAMOND

SIEMENS

nagarro

Atos

Trimble.

Raiffeisen BANK
Banking așa cum trebuie

SILVER

accenture

endava

cegeka

PARTNERS

coresi

<ScriuCod/>

CODECAMP_

MEDIA PARTNERS

CAPITAL

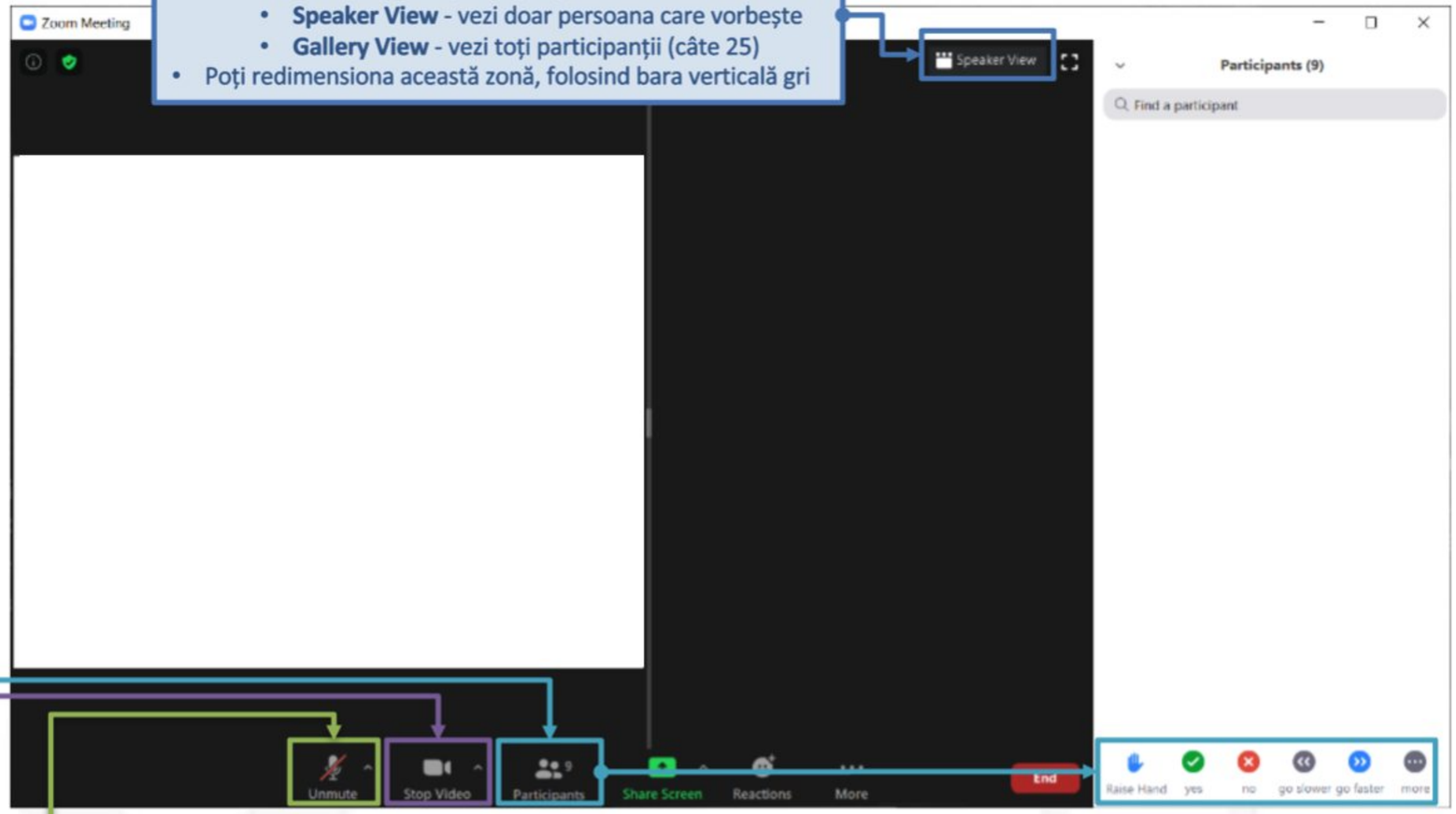
BizBrasov
Stirile care conteaza in Brasov

start#up

ROCK FM
It Rocks!

Pasul 4: Cum vezi cine vorbește?

- Ai 2 moduri de a vedea participanții:
  - **Speaker View** - vezi doar persoana care vorbește
  - **Gallery View** - vezi toți participanții (câte 25)
- Poți redimensiona această zonă, folosind bara verticală gri



Pasul 1: Conectează-te Audio

- Dacă nu ne auzi, folosește săgeţica din dreapta pentru a configura corect sunetul - alege **"Use computer sound"**
- Dacă ai configurat sunetul, dă un-mute la microfon și salută-ne ☺

Pasul 2: Pornește Camera Video

- Ne dorim o întâlnire cât mai **interactivă**, ca și când am fi în aceeași încăpere (55% din comunicare e non-verbală)
- Nu trebuie să fii la 4 ace

Pasul 3: Panoul de participanți și gesturi non-verbale

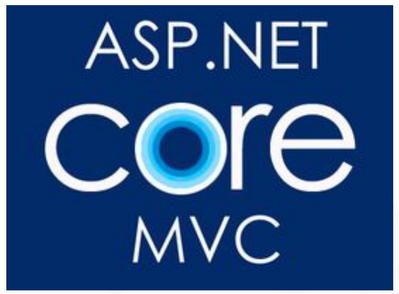
- Ai câteva gesturi pe care le poți folosi în timpul prezentării: **Raise Hand / Yes / No / Go slower / Go faster / Like / Dislike / Clap**
- Verifică **numele tău**, să reflecte realitatea. Te poți redenumi folosind meniul din dreapta numelui tău: "More > Rename"

Go to **www.menti.com** and use the code **4888 9854**

Agenda

- The modern .Net
- Does front-end mean JavaScript any more?
- The state of front-end development – what is Web Assembly?
- Exploring Blazor
- Exploring SignalR
- Short live demo developing a simple app faster than ever

Let’s get to business
Building SPA’s



Web

Build web apps and services for macOS, Windows, Linux, and Docker.

Mobile

Use a single codebase to build native mobile apps for iOS, Android, and Windows.

Desktop

Create beautiful and compelling desktop apps for Windows and macOS.

Microservices

Create independently deployable microservices that run on Docker containers.

Game Development

Develop 2D and 3D games for the most popular desktops, phones, and consoles.

Machine Learning

Add vision algorithms, speech processing, predictive models, and more to your apps.

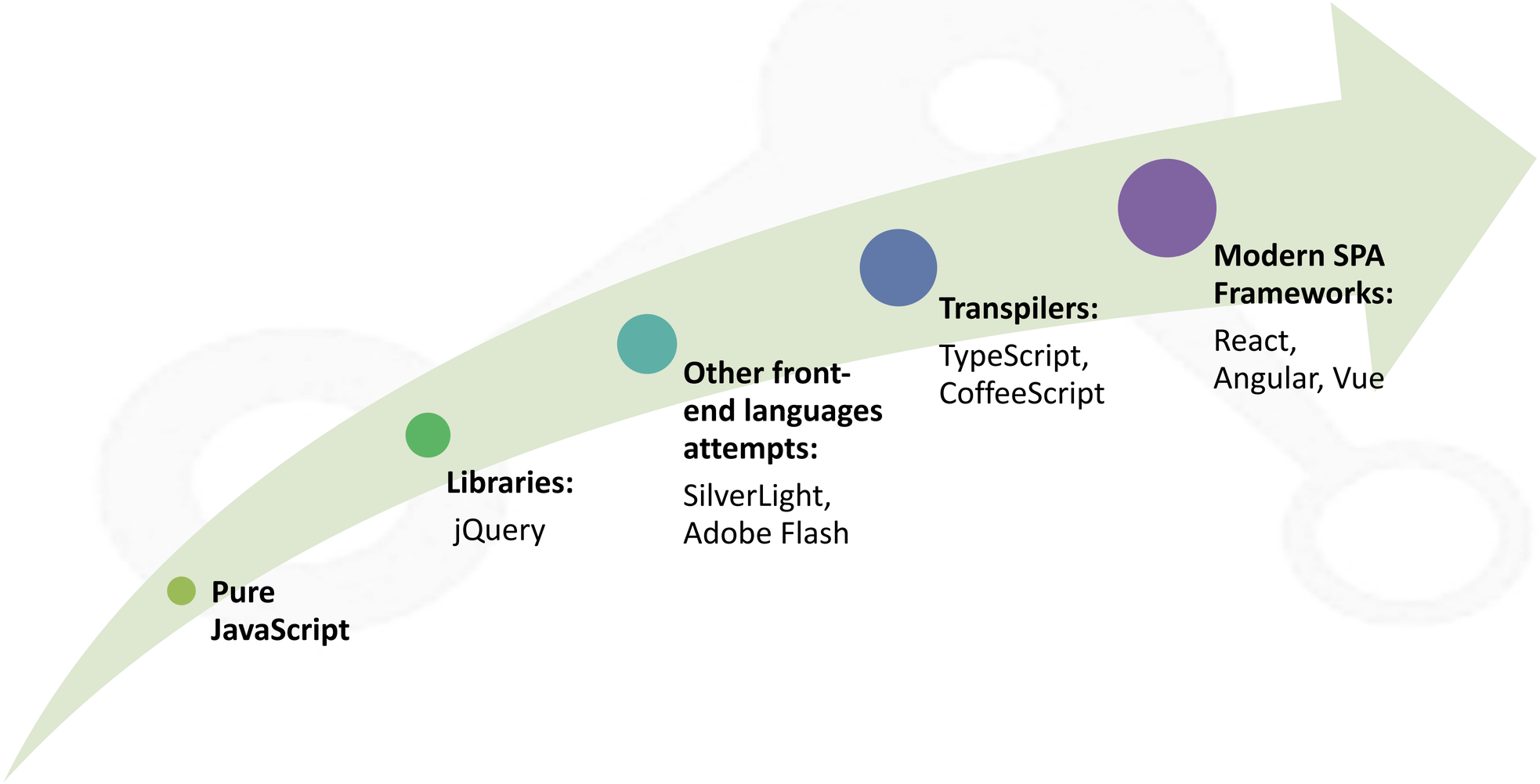
Cloud

Consume existing cloud services, or create and deploy your own.

Internet of Things

Make IoT apps, with native support for the Raspberry Pi and other single-board computers.

A history of front-end development



Web Assembly

What is Web Assembly?

Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications. (webassembly.org)

- **Efficient and fast (runs at native speeds)**
- **Supported by all major browsers (🦊, 🌐, 🧭, 🌐)**
- **A lot of languages already compile to WASM**
  - **C# - with our star today - Blazor 😁**
  - **C++ (with Emscripten, Rust, Kotlin, Swift and others)**

What is Blazor

- Blazor lets you build interactive web UIs using C# instead of JavaScript

- Blazor apps are composed of reusable web UI components implemented using C#, HTML, and CSS

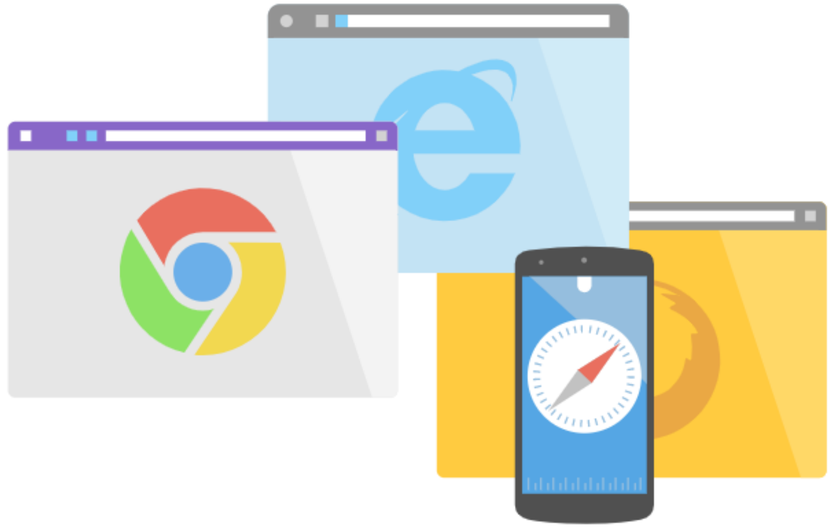
- Both client and server code is written in C#

Advantages

- All the benefits of a modern SPA framework: Reusable components, routing, dependency injection, 2-way binding

- Wide browser support, including mobile browsers

- Blazor was shipped with .Net Core 3.1 which is a LTS (long time support) version



Advantages

- Use a single language (C#) end-to-end - benefit from .NET's performance, reliability, and security

- Use the existing .NET ecosystem of .NET libraries

SignalR

- Library for building real-time apps that’s part of the ASP.NET framework

- Client SDKs for JavaScript, .NET and Java

- One of the fastest real-time frameworks around

SignalR

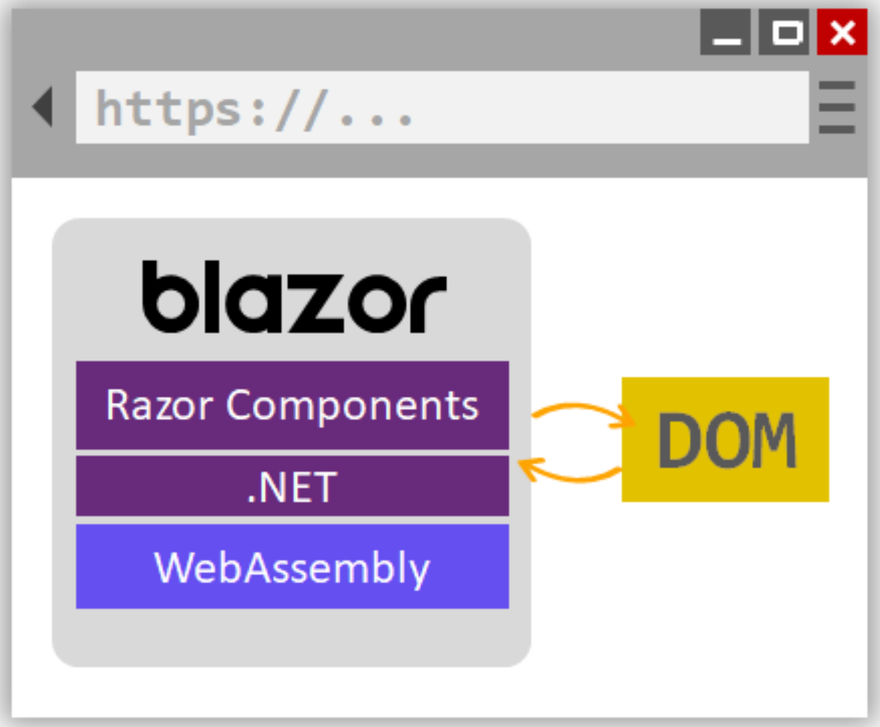
[How does it work?]

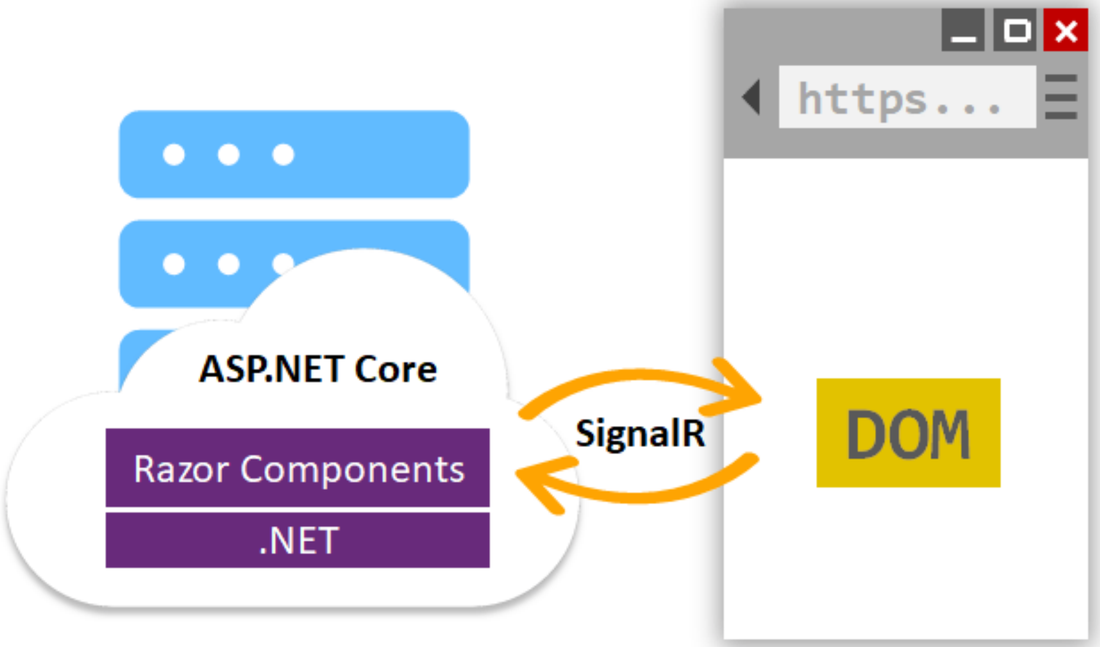
SignalR is an abstraction over some bi-directional communication protocols. It tries to pick the best/ modern protocol from top to bottom:

- WebSockets
- Server Sent Events
- Forever Frame
- Long Polling

```
dotnet add package Microsoft.AspNetCore.SignalR.Client
```

WebAssembly vs Server-side Blazor





Getting Started with Blazor

- Make sure you have installed the latest version of:
  - Visual Studio (Community Edition is free) or
  - Visual Studio Code or
  - Rider
- You can even try the preview version for the latest Blazor features
- When installing VS make sure you check the
  - Data storage and processing workload
  - ASP Net and Web development
  - .Net Core cross-platform development
- Make sure you have the latest .Net Core 5 SDK (normally installed with VS)

Hands-on

Let’s create a Blazor WebAssembly project from the CLI

Run the following on any OS:

dotnet new blazorwasm --name DemoWebAssembly --hosted --pwa --framework net5.0

Convinced yet?

A Blazor page

Routing

The page HTML
or the page layout
(Razor syntax that
you know and love)

The page behavior code
(only C# code)

```
@page "/counter"

<h1>Counter</h1>

<p>Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
<input type="text" @bind="currentCount" />

@code {
    int currentCount = 0;

    void IncrementCount()
    {
        currentCount++;
    }
}
```

A Blazor component

Components/ChildComponent.razor:

CSHTML

```
<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]
    public string Title { get; set; }

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

    [Parameter]
    public EventCallback<MouseEventArgs> OnClick { get; set; }
}
```

Pages/ParentComponent.razor:

CSHTML

```
@page "/ParentComponent"

<h1>Parent-child example</h1>

<ChildComponent Title="Panel Title from Parent"
                OnClick="@ShowMessage">
    Content of the child component is supplied
    by the parent component.
</ChildComponent>

<p><b>@messageText</b></p>

@code {
    private string messageText;

    private void ShowMessage(MouseEventArgs e)
    {
        messageText = $"     Blazor! ({e.ScreenX})";
    }
}
```

Routing

CSHTML

```
@page "/BlazorRoute"
@page "/DifferentBlazorRoute"

<h1>Blazor routing</h1>
```

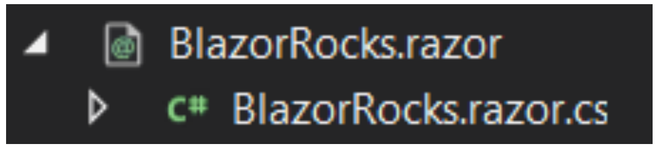
Multiple routes

CSHTML

```
@page "/RouteParameter"
@page "/RouteParameter/{text}"

<h1>Blazor is @Text!</h1>

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

    protected override void OnInitialized()
    {
        Text = Text ?? "fantastic";
    }
}
```

Route parameters

Separate file for the C# (behaviour) code



Or create a partial class with the same name

Pages/BlazorRocks.razor:

CSHTML

```
@page "/BlazorRocks"
@inherits BlazorRocksBase

<h1>@BlazorRocksText</h1>
```

BlazorRocksBase.cs:

C#

```
using Microsoft.AspNetCore.Components;

namespace BlazorSample
{
    public class BlazorRocksBase : ComponentBase
    {
        public string BlazorRocksText { get; set; } =
            "Blazor rocks the browser!";
    }
}
```

Component Lifecycle methods

```
protected override async Task OnInitializedAsync() { await ... }

protected override void OnInitialized() { ... }

public override async Task SetParametersAsync(ParameterView parameters) {    await ...
        await base.SetParametersAsync(parameters);
}

protected override async Task OnParametersSetAsync() { await... }

protected override void OnAfterRender(bool firstRender) {
        if (firstRender) { ... }
}

protected override bool ShouldRender() {
        return false;
}
```

Suppress component render

```
@implements IDisposable
@code {
        public void Dispose() { ... }
}
```

TIP: Calling StateHasChanged() notifies the component that its state has changed so it is rerendered.

Dependency Injection

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<IDataAccess, DataAccess>();
}
```

Add your service in Startup.cs

```
@page "/customer-list"
@using Services
@inject IDataAccess DataRepository

@if (Customers != null)
{
    <ul>
        @foreach (var customer in Customers)
        {
            <li>@customer.FirstName @customer.LastName</li>
        }
    </ul>
}

@code {
    private IReadOnlyList<Customer> Customers;

    protected override async Task OnInitializedAsync()
    {
        // The property DataRepository received an implementation
        // of IDataAccess through dependency injection. Use
        // DataRepository to obtain data from the server.
        Customers = await DataRepository.GetAllCustomersAsync();
    }
}
```

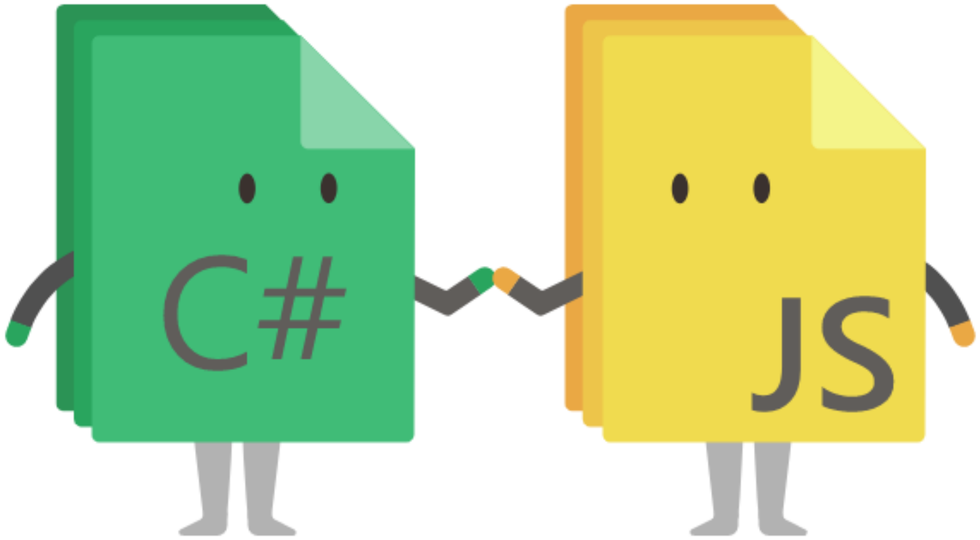
Forms and validation

```
using System.ComponentModel.DataAnnotations;

public class ExampleModel
{
    [Required]
    [StringLength(10, ErrorMessage = "Name is too long.")]
    public string Name { get; set; }
}
```

Input component	Rendered as…
InputText	<input>
InputTextArea	<textarea>
InputSelect	<select>
InputNumber	<input type="number">
InputCheckbox	<input type="checkbox">
InputDate	<input type="date">

```
<EditForm Model="@exampleModel" OnValidSubmit="@HandleValidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <InputText id="name" @bind-Value="exampleModel.Name" />

    <button type="submit">Submit</button>
</EditForm>

@code {
    private ExampleModel exampleModel = new ExampleModel();

    private void HandleValidSubmit()
    {
        Console.WriteLine("OnValidSubmit");
    }
}
```

JavaScript interop



Your C# code can easily call JavaScript APIs and libraries. You can continue to use the large ecosystem of JavaScript libraries that exist for client side UI while writing your logic in C#.

```
<script>
  window.handleTickerChanged = (symbol, price) => {
    // ... client-side processing/display code ...
  };
</script>
```

```
@inject IJSRuntime JSRuntime

@code {
    protected override void OnInitialized()
    {
        StocksService.OnStockTickerUpdated += stockUpdate =>
        {
            JSRuntime.InvokeVoidAsync("handleTickerChanged",
                stockUpdate.symbol, stockUpdate.price);
        };
    }
}
```

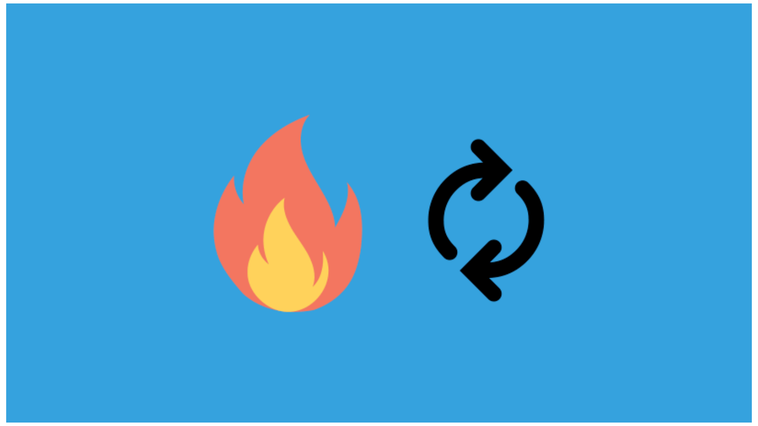
Hands-on

Let’s build a small BlazorServer App

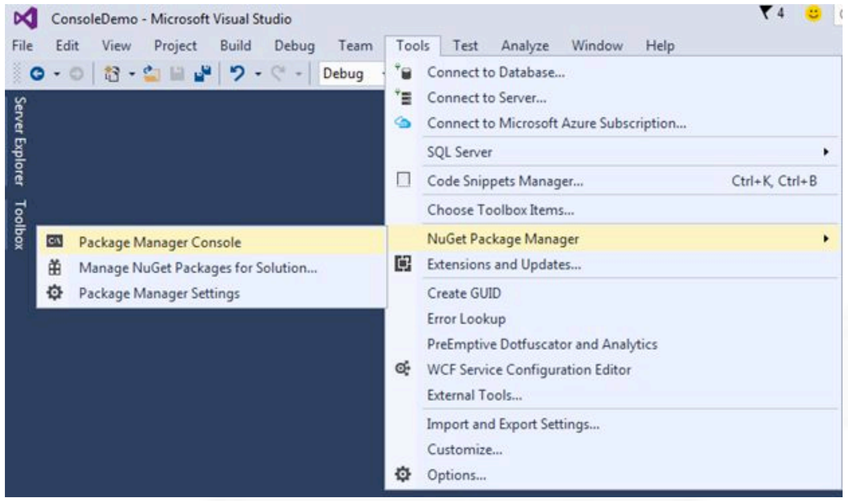
Run the following on any OS:

dotnet new blazorserver --name BlazorServerWeather --auth Individual --framework net5.0 [--use-local-db]

Let’s try the Hot Reload feature



Used commands



(Or any other terminal - run on project root directory)

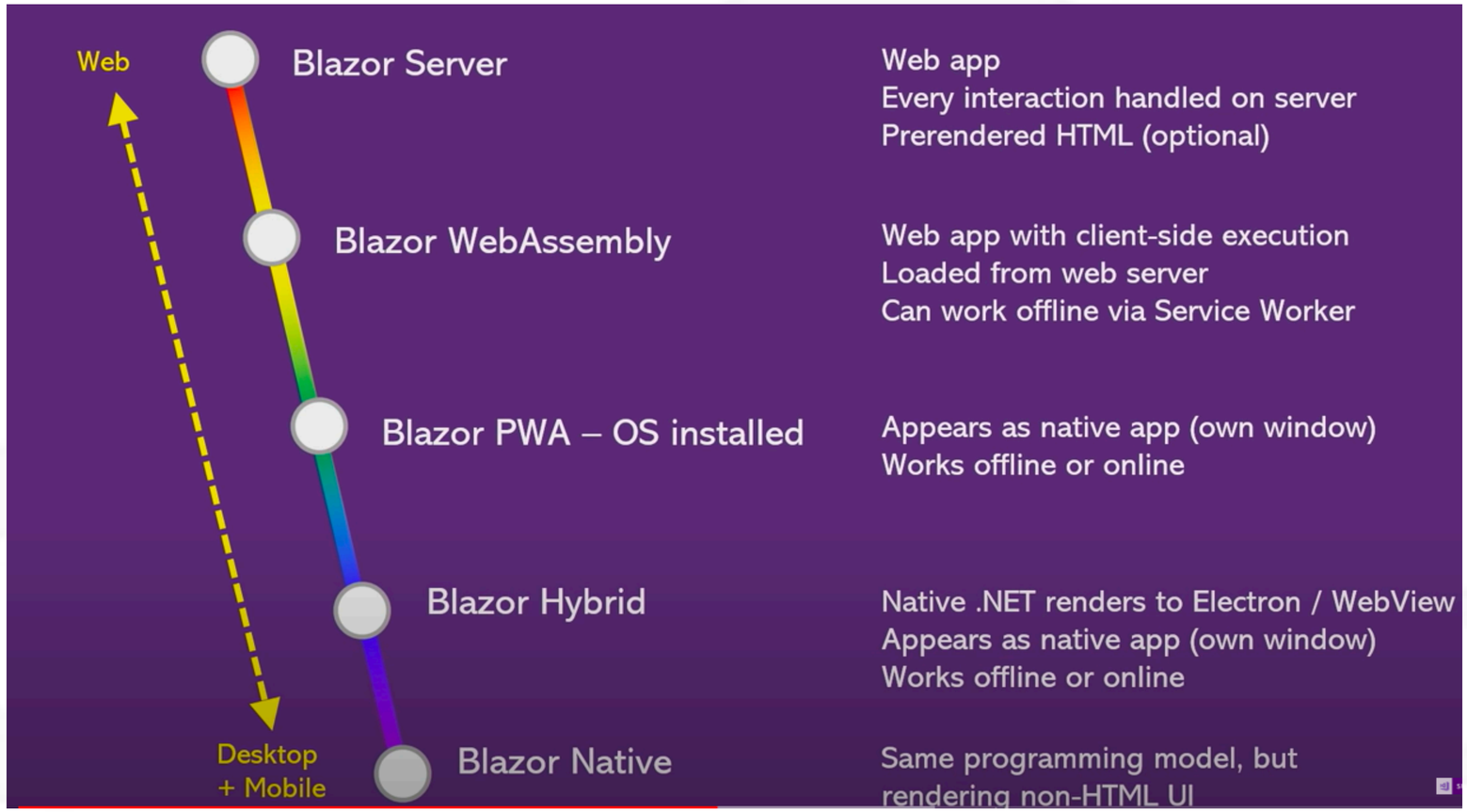
dotnet ef migrations add “MigrationName”

dotnet ef database update

dotnet run

dotnet watch

And there’s even more

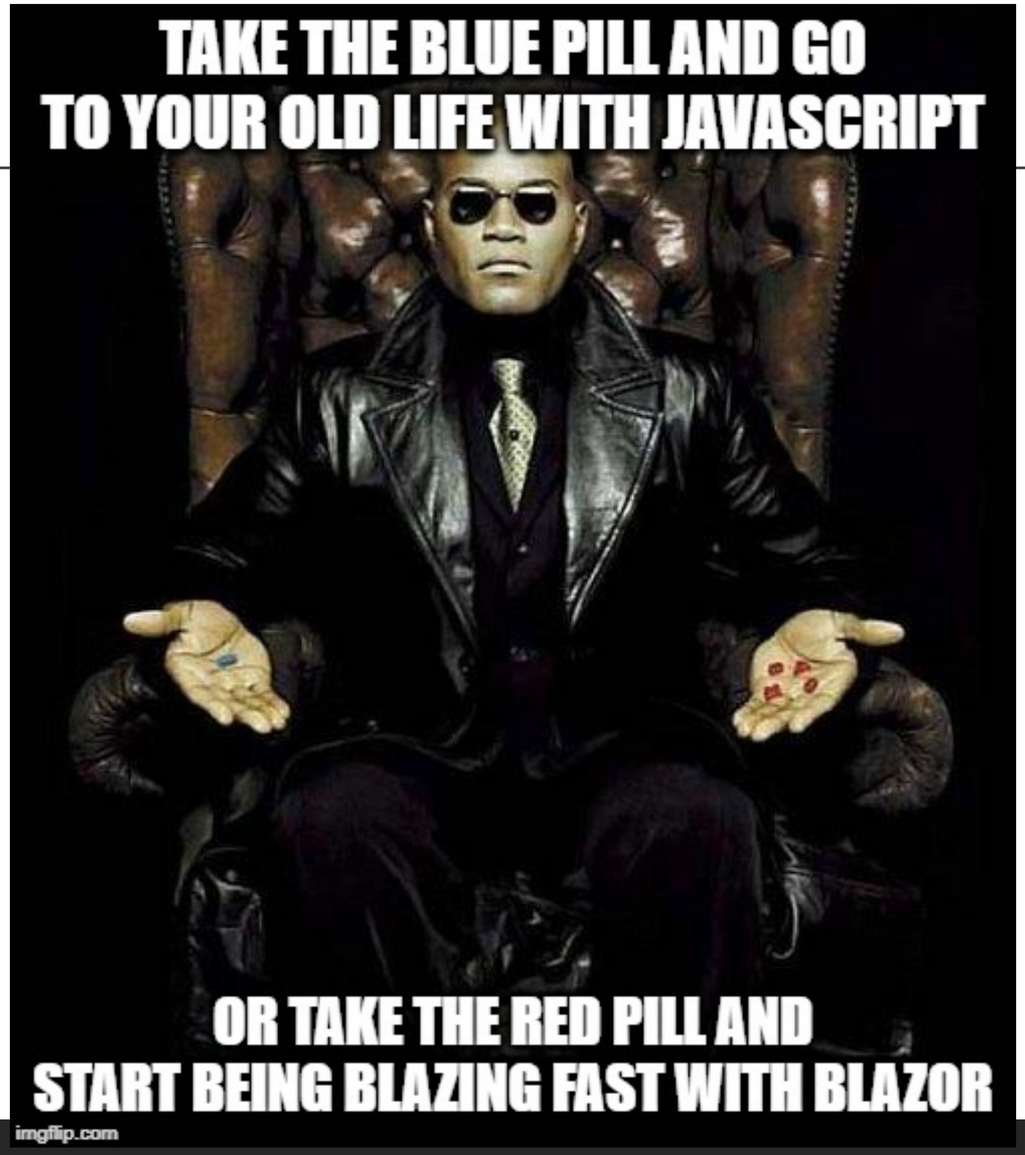


What is MAUI: https://docs.microsoft.com/en-us/dotnet/maui/what-is-maui

MAUI Hands-on: https://www.youtube.com/watch?v=Q0qLiBmbOTI

Blazor Mobile Hybrid Apps: https://docs.microsoft.com/en-us/mobile-blazor-bindings/

What will you choose?



Resources

Code and Presentation will be available at: https://github.com/AgileHubAssociation

SignalR with Blazor docs: https://docs.microsoft.com/en-us/aspnet/core/tutorials/signalr-blazor

Strongly Typed SignalR Hubs: https://docs.microsoft.com/en-us/aspnet/core/signalr/hubs

Awesome Blazor - Lots of cool resources: https://awesomeopensource.com/project/AdrienTorris/awesome-blazor

Free Blazor components: https://blazor.radzen.com/

MudBlazor - Free Material UI Blazor Implementation: https://mudblazor.com/

Blazorise - Free(mium) Bootstrap Blazor implementation: https://blazorise.com/docs/start/

Blazor documentation: https://docs.microsoft.com/en-us/aspnet/core/blazor/?view=aspnetcore-3.0

Blazor university: https://blazor-university.com/overview/what-is-blazor/

Commercial Blazor UI Frameworks: Telerik, DevExpress, Syncfusion, Infragistics, GrapeCity, jQWidgets.

Questions?

Feedback

http://bit.ly/peakit004-feedback

Completați acum

Durează 2-3 minute

Feedback anonim - pentru formator și AgileHub