dymaptic

Blazor for JavaScript Developers

Tim Purdum

DevUp Conf

August, 2025

techbash

esri Partner Network Gold

Esri Partner Conference 2023 Award Winner GIS for Good

Federal Small Business Specialty

ArcGIS Hub Specialty

Nonprofit Specialty

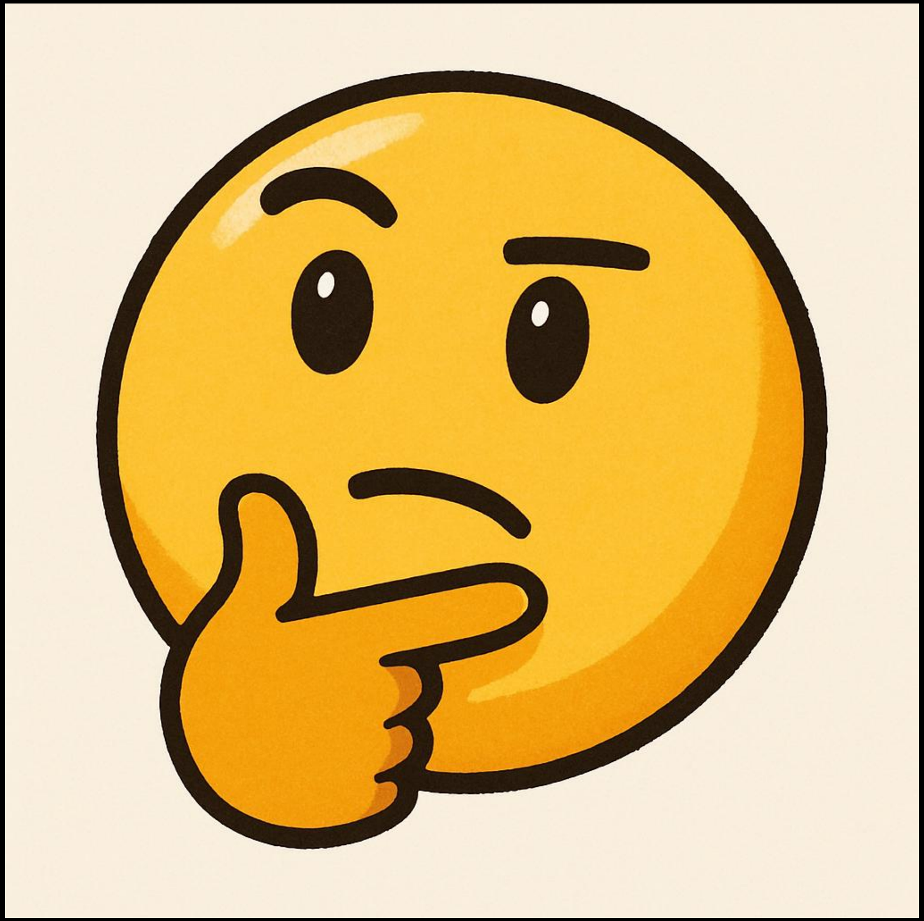
State and Local Government Specialty

WOMEN OWNED

SBA EDWOSB CERTIFIED

techbash

Why Did You Come to This Session?



techbash

Goals of the Session

- *Why* use Blazor
- Getting started
- Unique features and functionality
- Compare & contrast to JavaScript frameworks
- Pitfalls and drawbacks
- How it ties into the Asp.NET Core back-end Ecosystem
- How you can write interop code between Blazor and JS

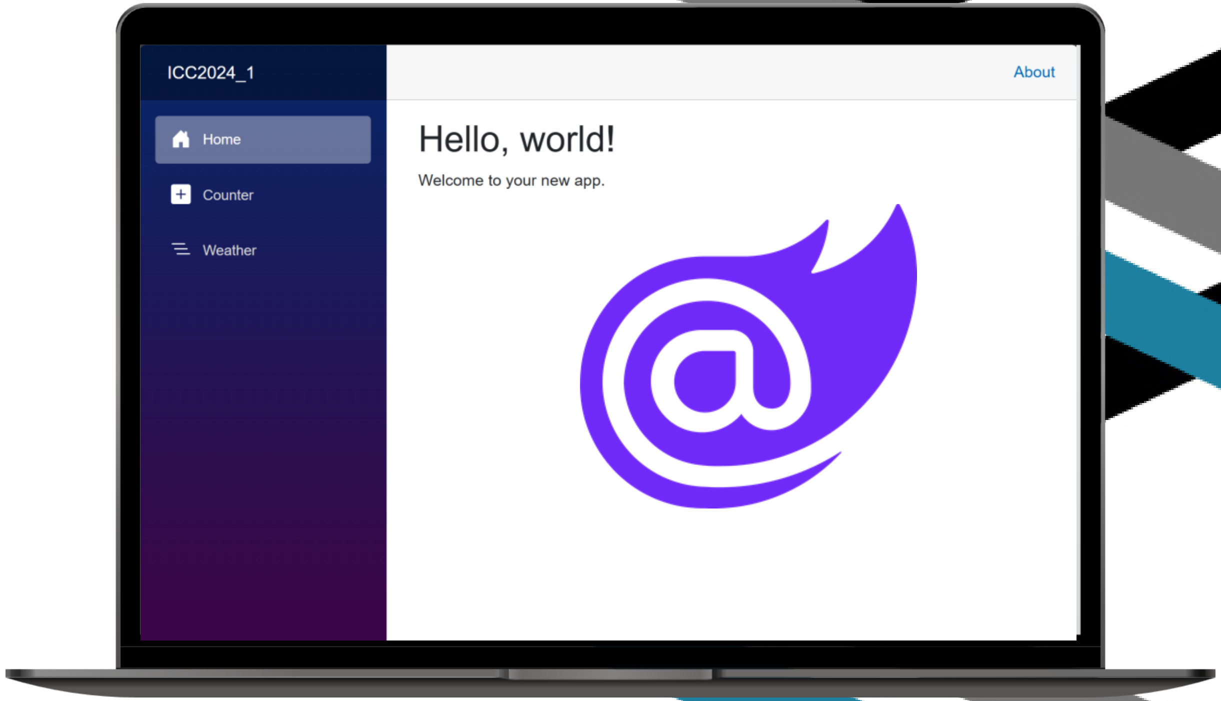
Why Use Blazor?

- Single language, full stack
- Shared data models
- Strongly-typed language
- Utilize existing developer skillsets
- Expanding an existing Asp.NET Core application
- Unique, server-first rendering modes unavailable in JS
- Shared code with .NET MAUI for desktop/mobile

techbash

What is Blazor?

- Modern full-stack web framework
- Built on Asp.NET Core and Modern .NET
- Released with.NET Core 3.1 in 2018
- Static and dynamic Server-Side rendering
- Client WebAssembly SPA applications or individual components
- High productivity with a single unifying language and framework
- Hot-reload == rapid development with robust dev tools



Hello, world!

Welcome to your new app.

techbash

Modern .NET & C#

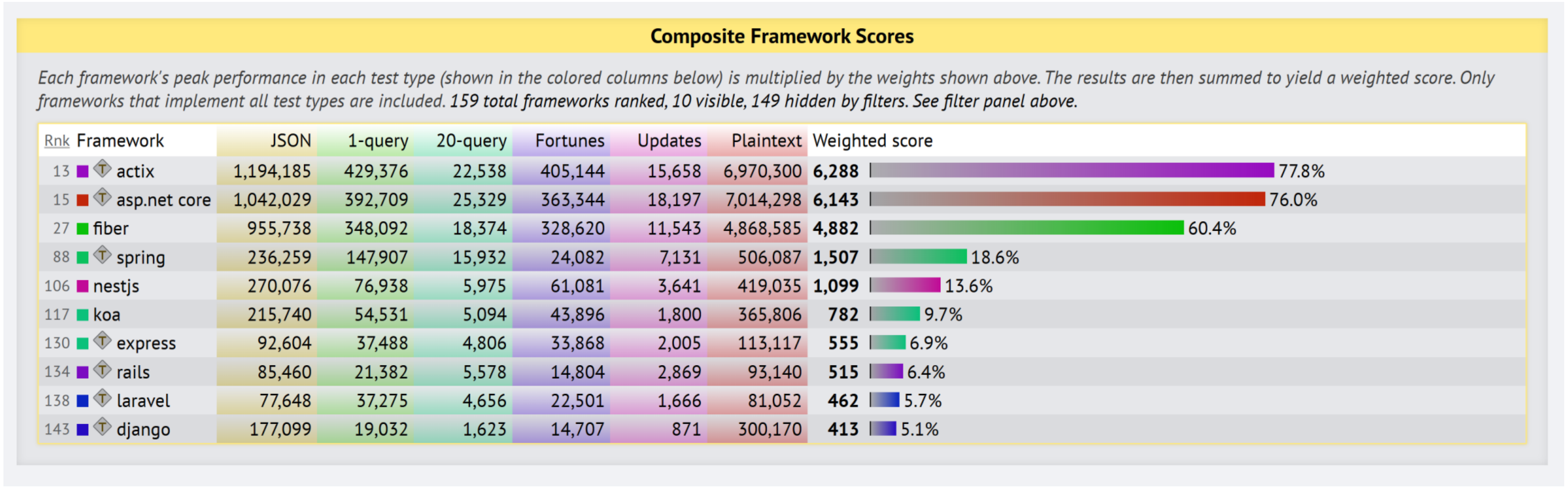
- **Rewritten from scratch with .NET Core in 2016**
- **Open-source, code available on GitHub**
- **.NET 5 2020+ Yearly release cycle**
- **Roslyn compiler runs C# to compile C#**
- **Runs on Windows, Mac, Linux, iOS, Android**
- **Scripts, web services, websites, mobile and desktop applications**
- **Robust Nuget package ecosystem**
- **Performance focus**
- **Both JIT and AOT Compilers**

techbash

Asp.NET Core Performance

Round 22 results

TechEmpower Framework Benchmarks Oct. 2023



Composite Framework Scores

Each framework's peak performance in each test type (shown in the colored columns below) is multiplied by the weights shown above. The results are then summed to yield a weighted score. Only frameworks that implement all test types are included. 159 total frameworks ranked, 10 visible, 149 hidden by filters. See filter panel above.

Rnk	Framework	JSON	1-query	20-query	Fortunes	Updates	Plaintext	Weighted score	
13	🟪 actix	1,194,185	429,376	22,538	405,144	15,658	6,970,300	**6,288**	77.8%
15	🟥 asp.net core	1,042,029	392,709	25,329	363,344	18,197	7,014,298	**6,143**	76.0%
27	🟩 fiber	955,738	348,092	18,374	328,620	11,543	4,868,585	**4,882**	60.4%
88	🟩 spring	236,259	147,907	15,932	24,082	7,131	506,087	**1,507**	18.6%
106	🟪 nestjs	270,076	76,938	5,975	61,081	3,641	419,035	**1,099**	13.6%
117	🟩 koa	215,740	54,531	5,094	43,896	1,800	365,806	**782**	9.7%
130	🟩 express	92,604	37,488	4,806	33,868	2,005	113,117	**555**	6.9%
134	🟪 rails	85,460	21,382	5,578	14,804	2,869	93,140	**515**	6.4%
138	🟦 laravel	77,648	37,275	4,656	22,501	1,666	81,052	**462**	5.7%
143	🟦 django	177,099	19,032	1,623	14,707	871	300,170	**413**	5.1%

techbash

Comparing Blazor to JavaScript Frameworks

	Angular	React	Vue	Blazor
Year Released	2010	2013	2014	2018
Created By	Google	Facebook (Meta)	Evan You (former Google empl.)	Microsoft
Language	TypeScript	JavaScript/TypeScript	JavaScript/TypeScript	C#/WebAssembly

techbash

Comparing Blazor to JavaScript Frameworks

Angular	React	Vue	Blazor
Templating	Templating	Templating	Templating
HTML Templates	JSX	Single-file Components	Razor Files
Code Injection	Code Injection	Code Injection	Code Injection
{{codeInjection}}	{codeInjection}	{codeInjection}	@codeInjection
[src]="propBinding"	src={propBinding}	:src={propBinding}	src="@propBinding"
(click)="jsFunction()"	onClick={jsFunction}	@click="jsfunction"	@onclick="csMethod"
2-Way Binding	2-Way Binding	2-Way Binding	2-Way Binding
Install @angular/forms	value={item}	v-model="item"	@bind="item" OR
[(ngModel)]="model.item"	onChange={setItem}		@bind:get="item" &
			@bind:set="method"

techbash

Comparing Blazor to JavaScript Frameworks

Angular

Project Structure

- src
  - app
    - app.component.css
    - app.component.html
    - app.component.ts
    - app.config.ts
    - app.module.ts
    - app.routes.ts
    - components
      - component1.css
      - component1.html
      - component1.ts
  - index.html
  - main.ts
  - styles.css

React (Next)

Project Structure

- public
- src
  - app
    - layout.tsx
    - page.tsx
    - routes.ts
    - components
      - comp1.tsx
      - comp1.module.css
    - pages
      - about
        - page.tsx
        - page.module.css
      - contact
        - page.txs
        - page.module.css
    - services
      - services1.ts

Vue

Project Structure

- public
- src
  - assets
    - main.css
    - logo.svg
  - components
    - Component1.vue
    - Component2.vue
  - router
    - index.ts
  - stores
    - counter.ts
  - views
    - HomeView.vue

Blazor

Project Structure

- src
  - App
    - App.csproj
    - Components
      - App.razor
      - Layout
        - MainLayout.razor
      - Pages
        - About.razor
      - Routes.razor
    - wwwroot
      - app.css
  - App.Client
    - App.Client.csproj
    - Components
      - ClientComp1.razor
    - Pages
      - Home.razor

Comparing Blazor to JavaScript Frameworks

Angular

- Single-Page Application
- Client-Only
- Requires a Back-end Paired Technology

React (Next)

- React *was* Client-first SPA
- NextJS is moving towards Server Components as the default
- Server Components have direct access to data stores, but no direct user interactivity like event handlers
- Many options of routers, state management frameworks, plugins

Vue

- Single-Page Application
- Site and components *can* be pre-rendered on the server

Blazor

- Only truly Client *AND* Server interactive framework with Server Components using event handlers over Web Sockets
- That extra flexibility also adds some complexity – it is important to know where your component is running and understand the render modes
- C# can only run in the browser when compiled to WebAssembly, which must still communicate with the DOM via JavaScript
- Web Sockets and WebAssembly cannot compete with JavaScript for high-performance real-time interactions (e.g., gaming)

techbash

Getting Started

- Get .NET
  - Download from https://dotnet.microsoft.com
  - winget install Microsoft.DotNet.SDK.9
- Get Language Support
  - VS Code + VS License – C# Dev Kit Extension
  - Visual Studio - Community Edition (free), Professional, Enterprise
  - JetBrains Rider – Free Community License, Paid Prof. License

techbash

Getting Started (2)

- `dotnet new list` – shows all the available templates
- `dotnet new blazor -h` – displays help for completions
- `dotnet new blazor -int Auto -au Individual -o HelloWorld`
  - `-int Auto` – auto interactive mode
  - `-au Individual` – adds individual authentication user accounts
  - `-o HelloWorld` – sets the name and output folder of the project
- `cd HelloWorld` – navigate
- `code .` – open the folder for editing
- `dotnet run -lp https` – run the application!

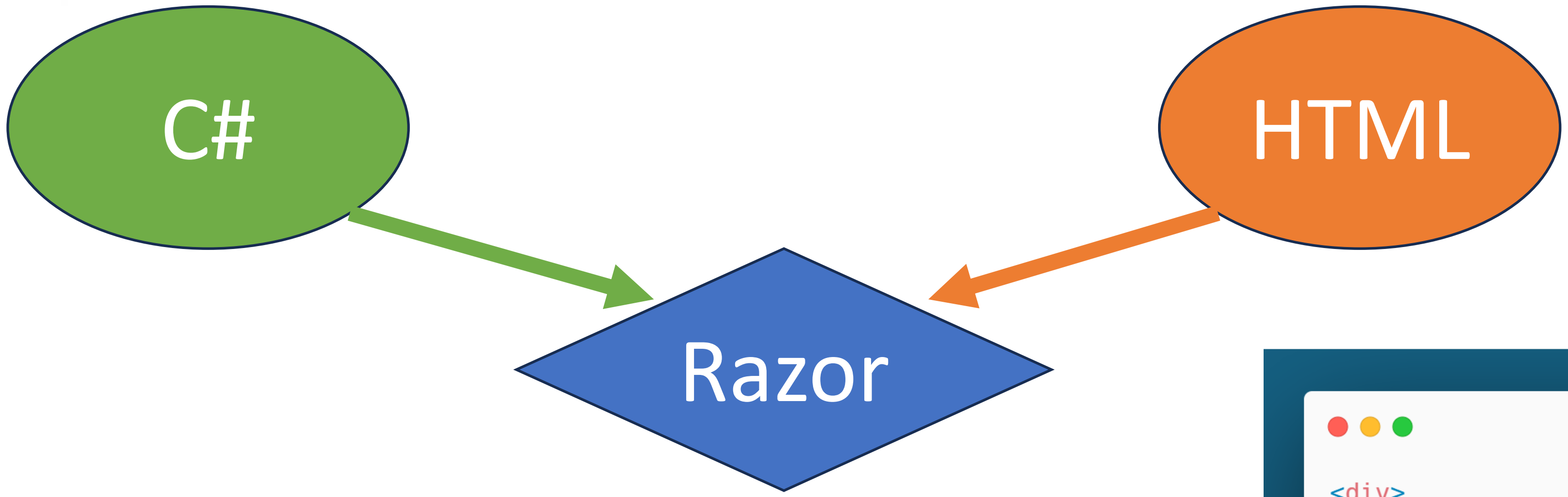
DON'T PANIC

techbash

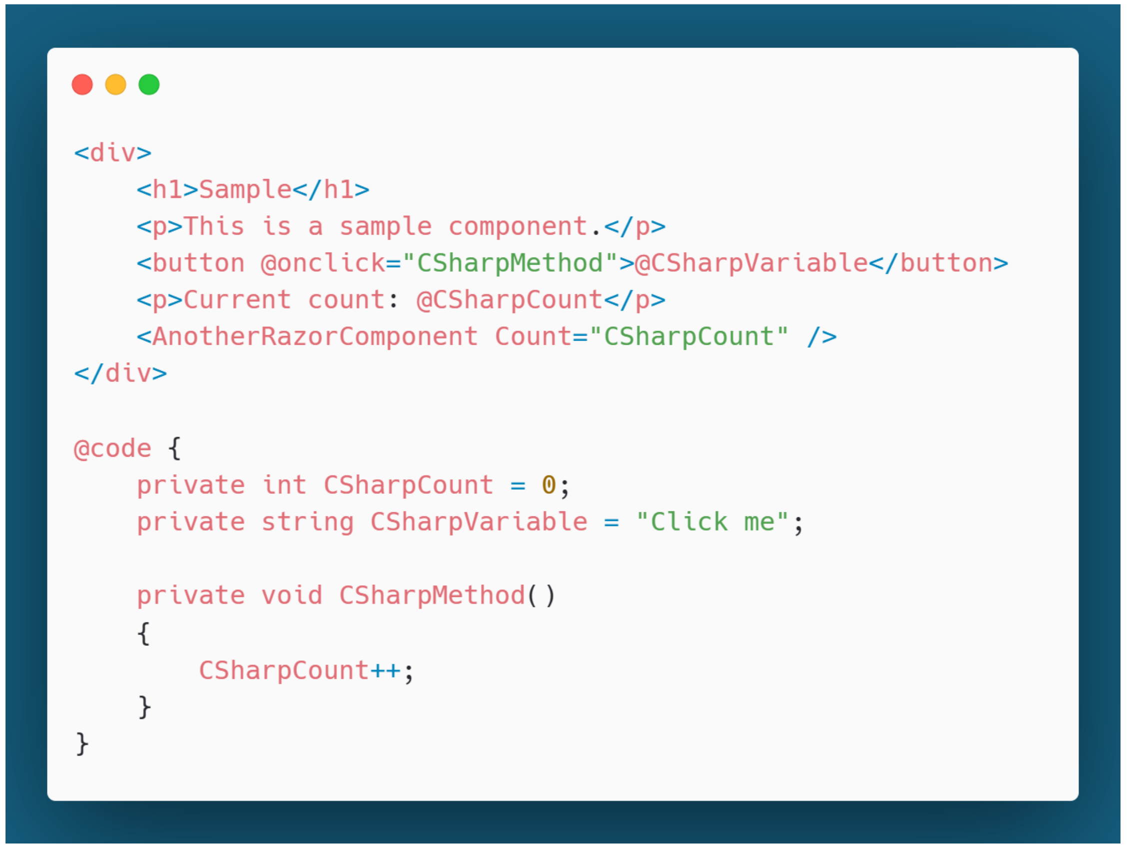
Blazor Supports Modern Web Standards

- HTML
  - Full support, only changes would be to escape `@` text characters
  - Can create nested components primarily out of HTML simply for organizational structure, even if not using other functionality
  - Declarative head tags, links, scripts, and component-level injection of extra head content, links, and scripts
- CSS
  - Inline support
  - ***/wwwroot*** public asset folder
  - Import with link tags
  - *Scoped* CSS files per component: e.g., `Component1.razor` => `Component1.razor.css`
- JavaScript
  - Script tag support
  - Module support
  - Interop calls from C# via `IJSRuntime` in Interactive Render modes

Razor (the Syntax and Components behind Blazor)



- `.razor` file extension
- Encapsulate UI and functionality
- Reusable and composable
- Each one can run client-side or server-side
- @ symbol identifies start of C# code
- Parentheses and Braces define code scopes
- Markup tags can be nested inside conditional logic on new lines

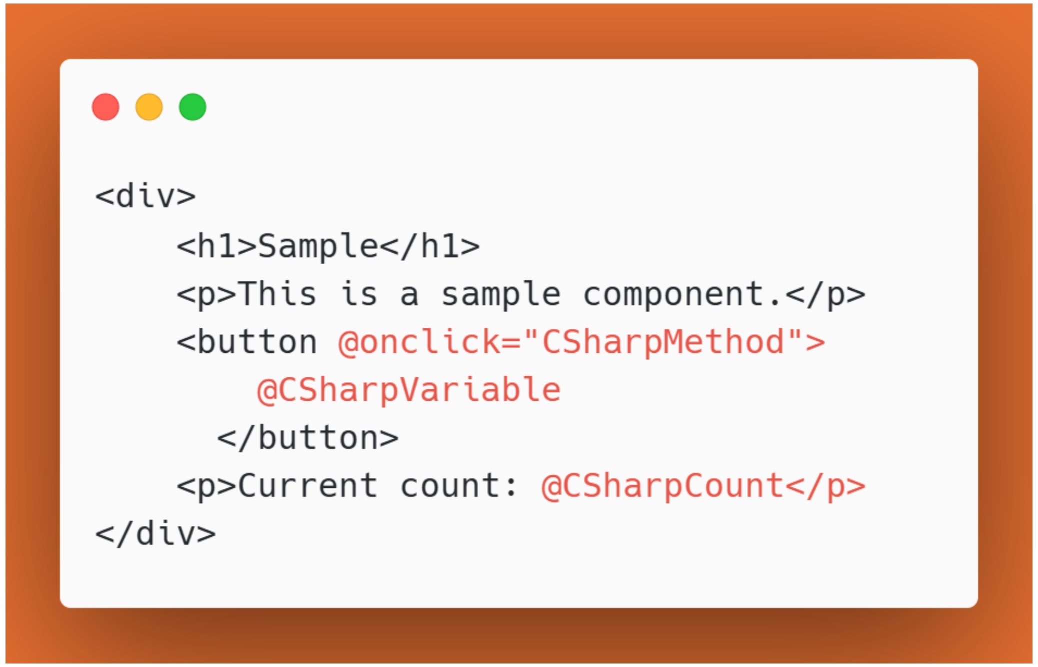


```
<div>
    <h1>Sample</h1>
    <p>This is a sample component.</p>
    <button @onclick="CSharpMethod">@CSharpVariable</button>
    <p>Current count: @CSharpCount</p>
    <AnotherRazorComponent Count="CSharpCount" />
</div>

@code {
    private int CSharpCount = 0;
    private string CSharpVariable = "Click me";

    private void CSharpMethod()
    {
        CSharpCount++;
    }
}
```

techbash

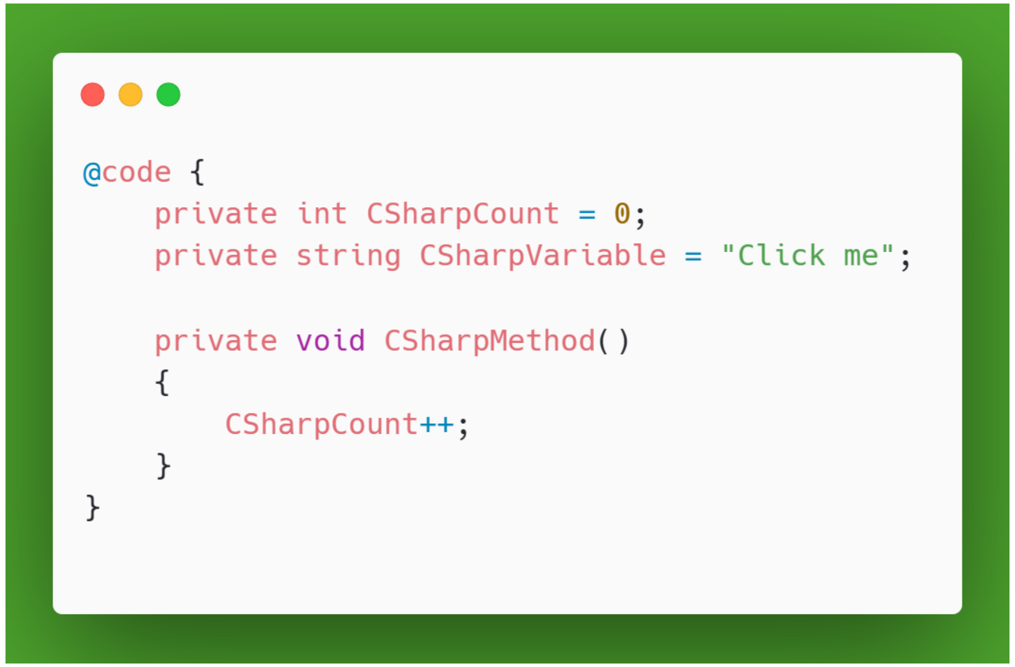
Razor Component Structure

Razor Markup



```
<div>
    <h1>Sample</h1>
    <p>This is a sample component.</p>
    <button @onclick="CSharpMethod">
        @CSharpVariable
      </button>
    <p>Current count: @CSharpCount</p>
</div>
```

Code Block

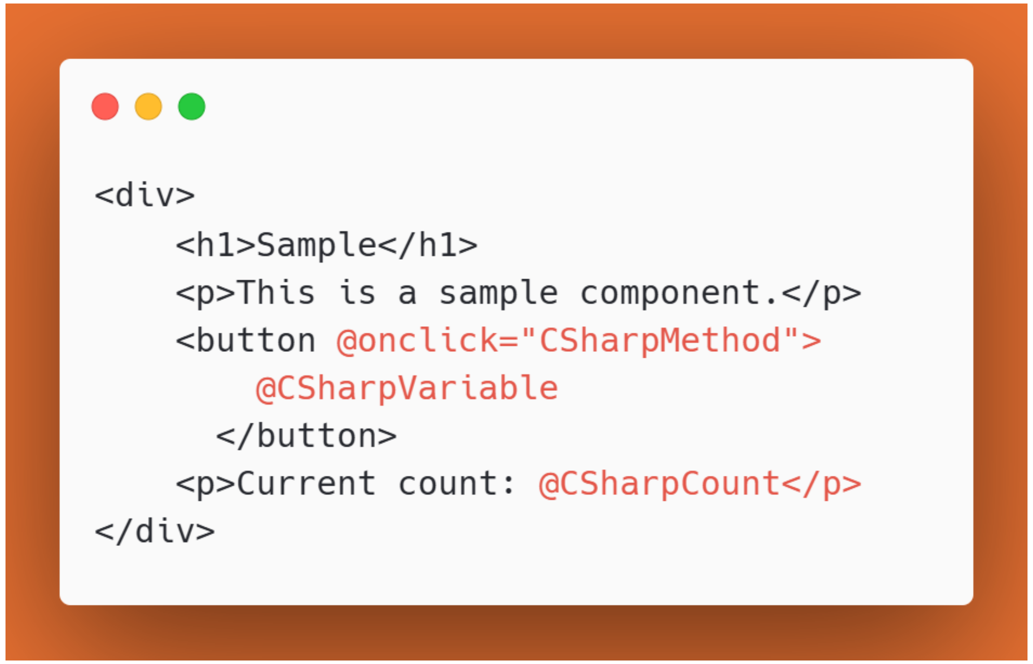


```
@code {
    private int CSharpCount = 0;
    private string CSharpVariable = "Click me";

    private void CSharpMethod()
    {
        CSharpCount++;
    }
}
```

techbash

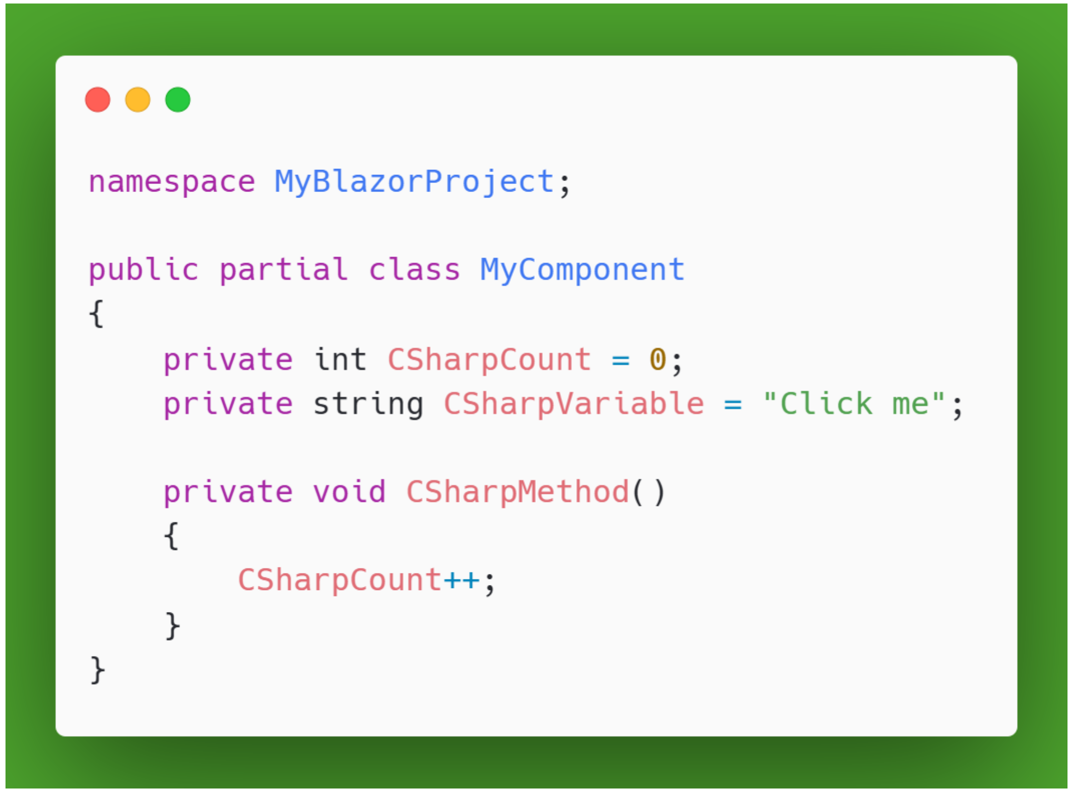
Razor Partial Class ("Code-Behind" Pattern)

MyComponent.razor



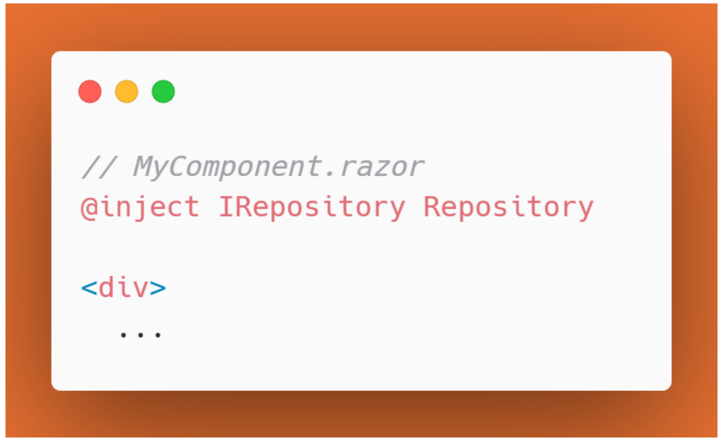
```
<div>
    <h1>Sample</h1>
    <p>This is a sample component.</p>
    <button @onclick="CSharpMethod">
        @CSharpVariable
      </button>
    <p>Current count: @CSharpCount</p>
</div>
```

MyComponent.razor.cs

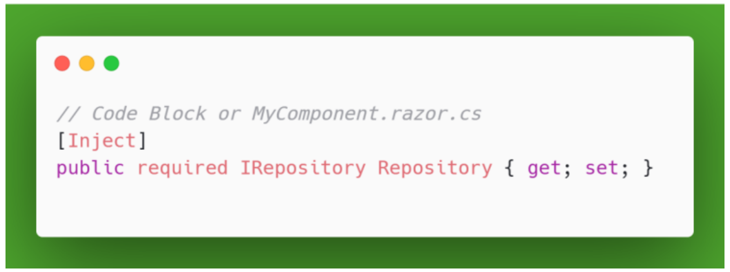


```
namespace MyBlazorProject;

public partial class MyComponent
{
    private int CSharpCount = 0;
    private string CSharpVariable = "Click me";

    private void CSharpMethod()
    {
        CSharpCount++;
    }
}
```

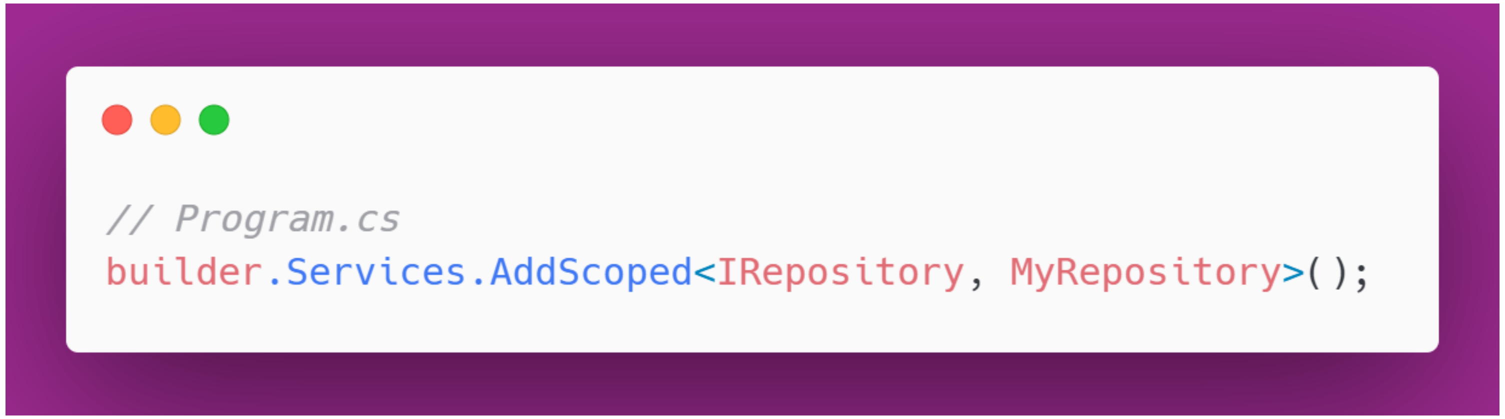
Dependency Injection



```
// MyComponent.razor
@inject IRepository Repository

<div>
  ...
```



```
// Code Block or MyComponent.razor.cs
[Inject]
public required IRepository Repository { get; set; }
```



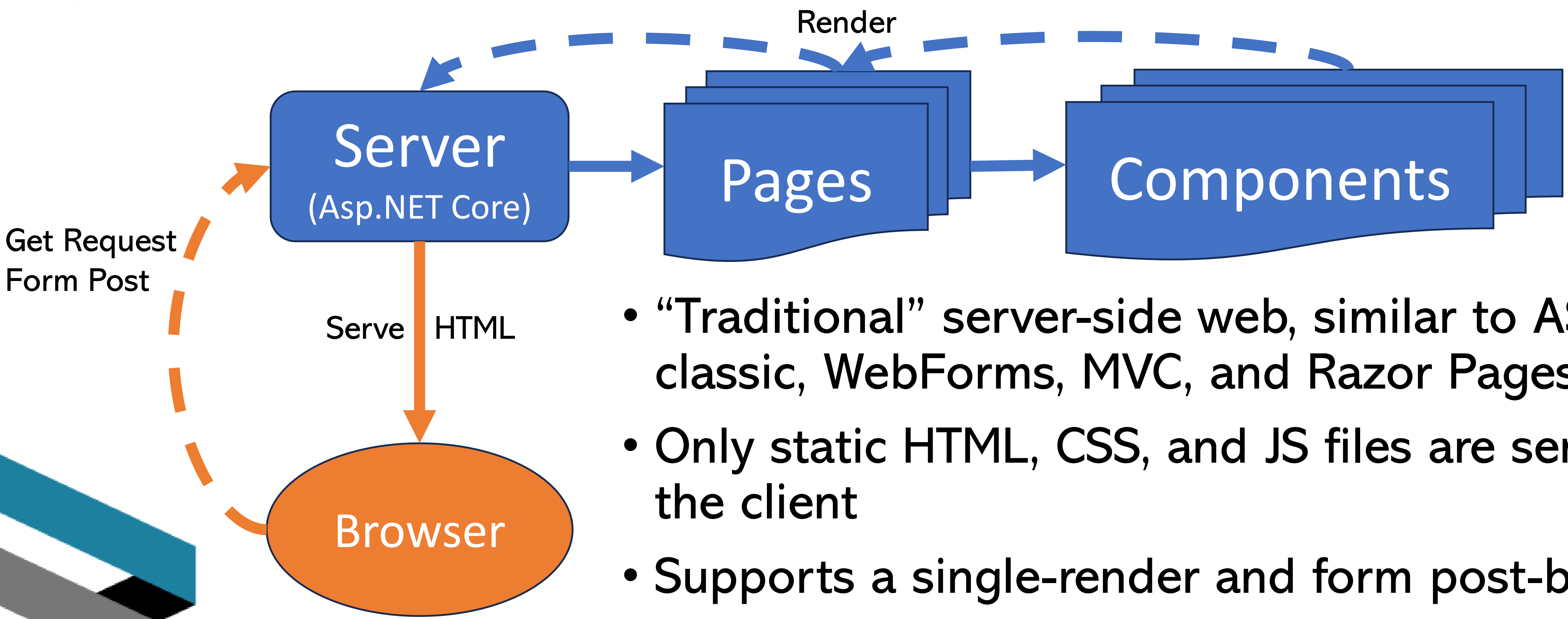
```
// Program.cs
builder.Services.AddScoped<IRepository, MyRepository>();
```

techbash

Blazor Component Render Modes

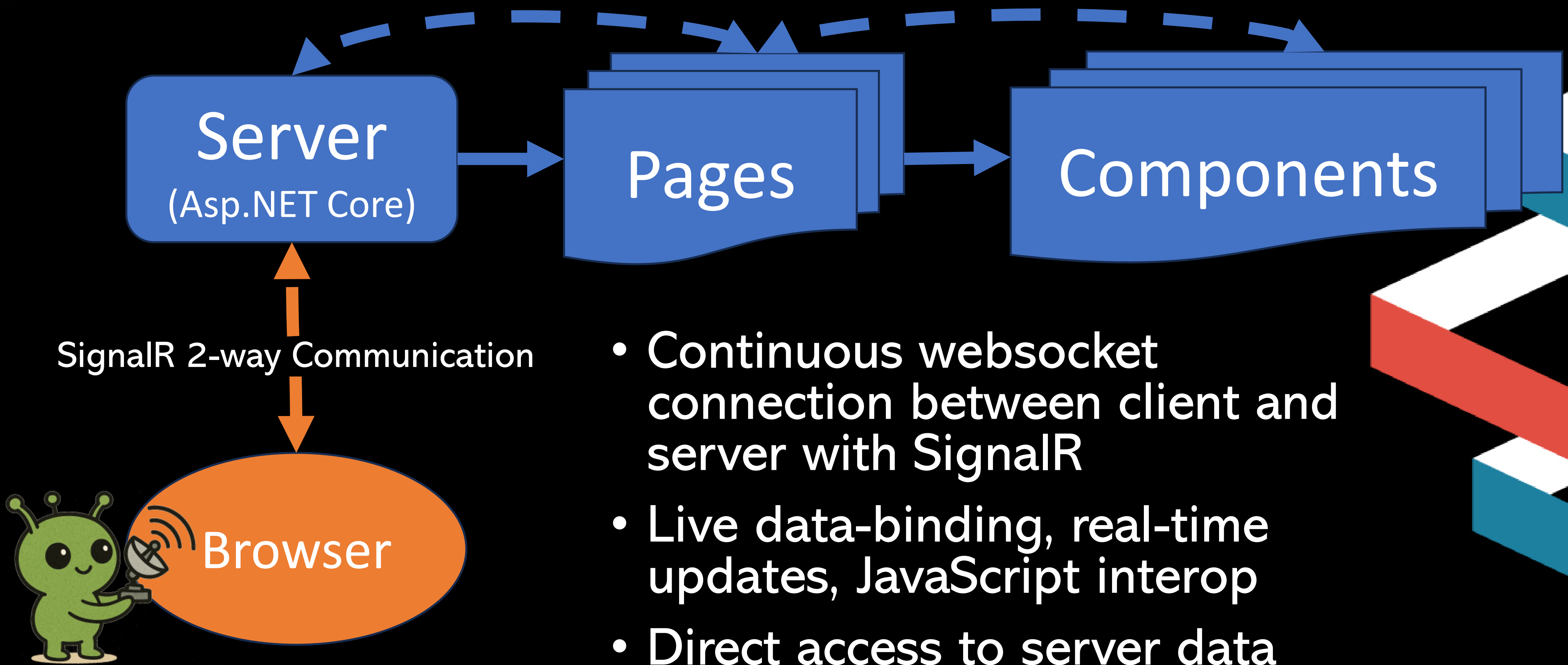
- Static Server Mode
- Interactive Server Mode
- Interactive WebAssembly Mode
- Interactive Auto Mode
- Blazor Hybrid (MAUI)

Blazor Render Modes: Static Server



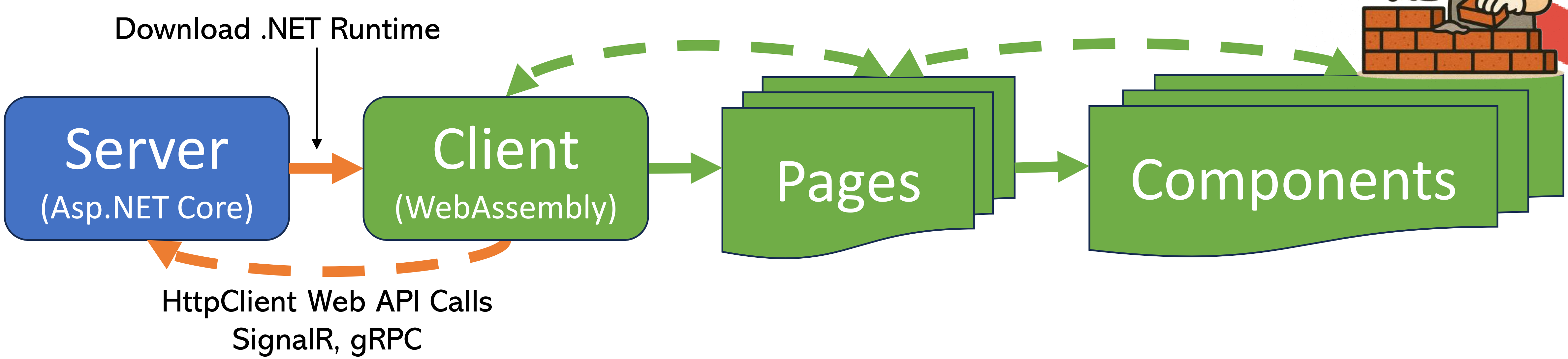
- "Traditional" server-side web, similar to ASP classic, WebForms, MVC, and Razor Pages
- Only static HTML, CSS, and JS files are sent to the client
- Supports a single-render and form post-backs
- No interactive updates via C# (can still use JS)
- Great for blog posts or other readonly content and simple forms

Blazor Render Modes: Interactive Server



- Continuous websocket connection between client and server with SignalR
- Live data-binding, real-time updates, JavaScript interop
- Direct access to server data store
- Fast on first load
- Leaving browser tabs open can cause disconnection issues

Blazor Render Modes: Interactive WebAssembly



- Runs in the client browser
- Live data-binding, real-time updates, JavaScript interop
- HttpClient calls to communicate with server web API

- Single-threaded
- Large/slow first load
- Fast interactions after load
- Closest in approach to most JS SPA frameworks

Blazor Render Modes: Interactive Auto

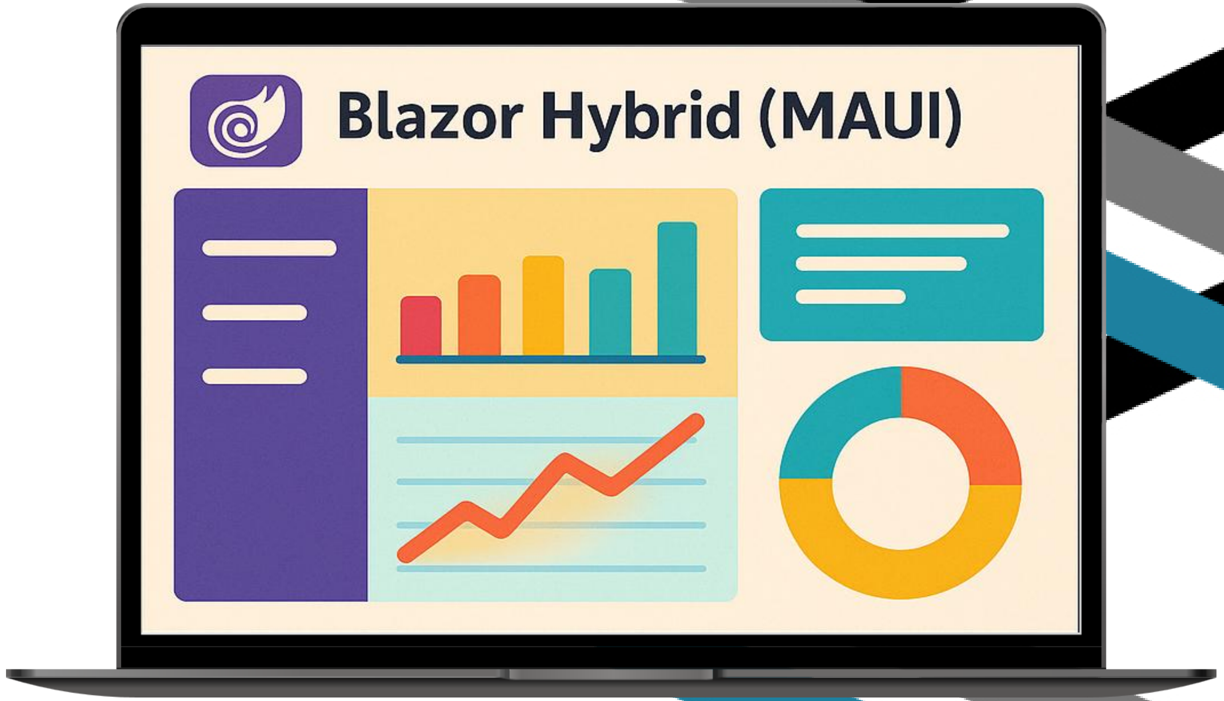
- On first load, runs from server, creating SignalR connection
- In the background, downloads .NET runtime and client code
- On next load, switches to running from WebAssembly
- “Best of both worlds”
  - Fast start on first load (server)
  - More responsive and robust interactions (client)
- Requires flexible data handling/abstraction to handle both client and server modes

techbash

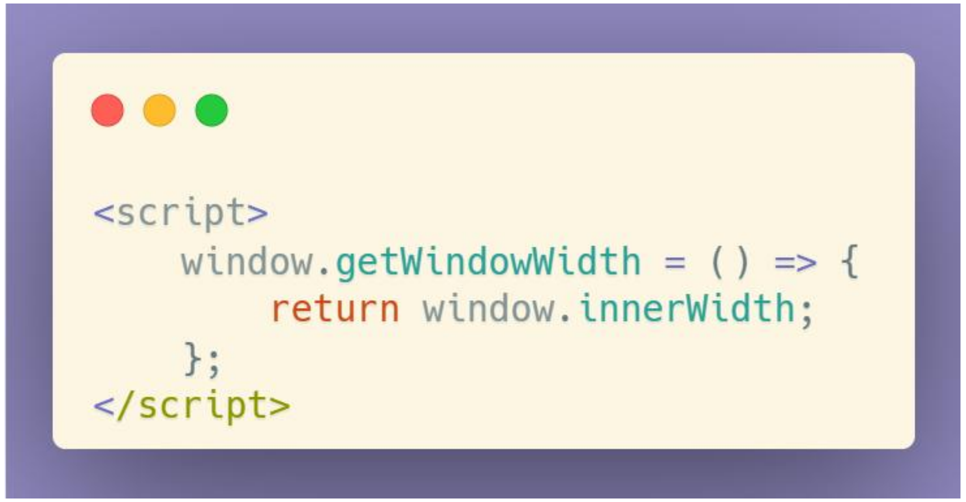
techbash

Blazor Hybrid (MAUI)

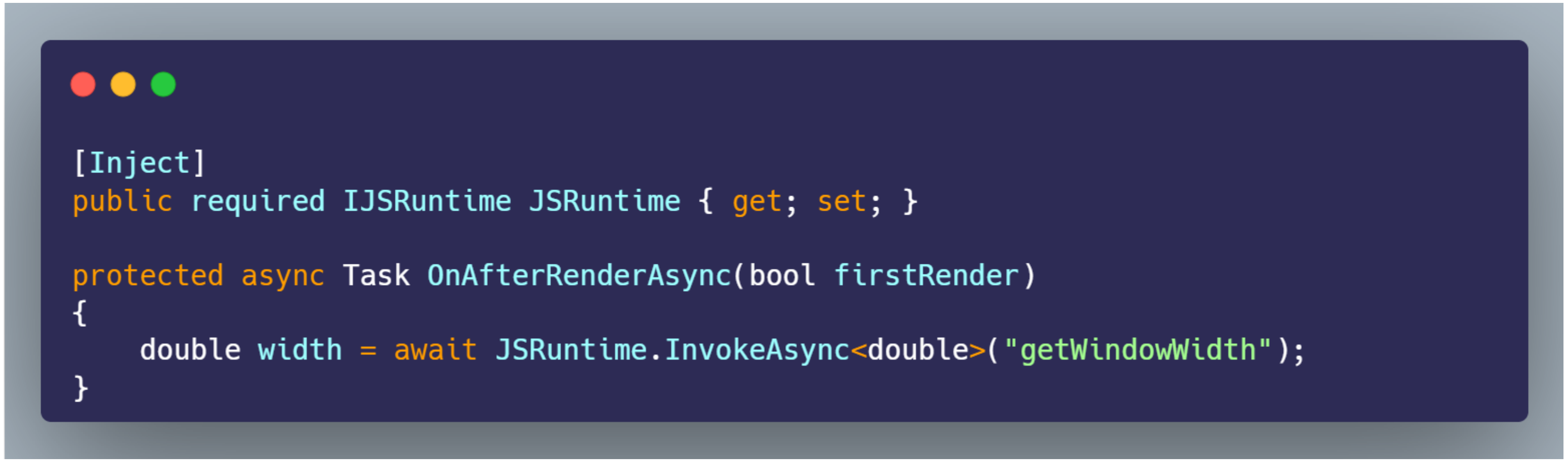
- Runs in a WebView in .NET MAUI (iOS, Android, Mac, Windows)
- Native .NET multi-threaded code execution (not WebAssembly)
- Access to device APIs (GPS, Bluetooth, photos, etc.)
- Can reuse components or entire UI applications between web, desktop, and mobile



Escaping Back into JavaScript…



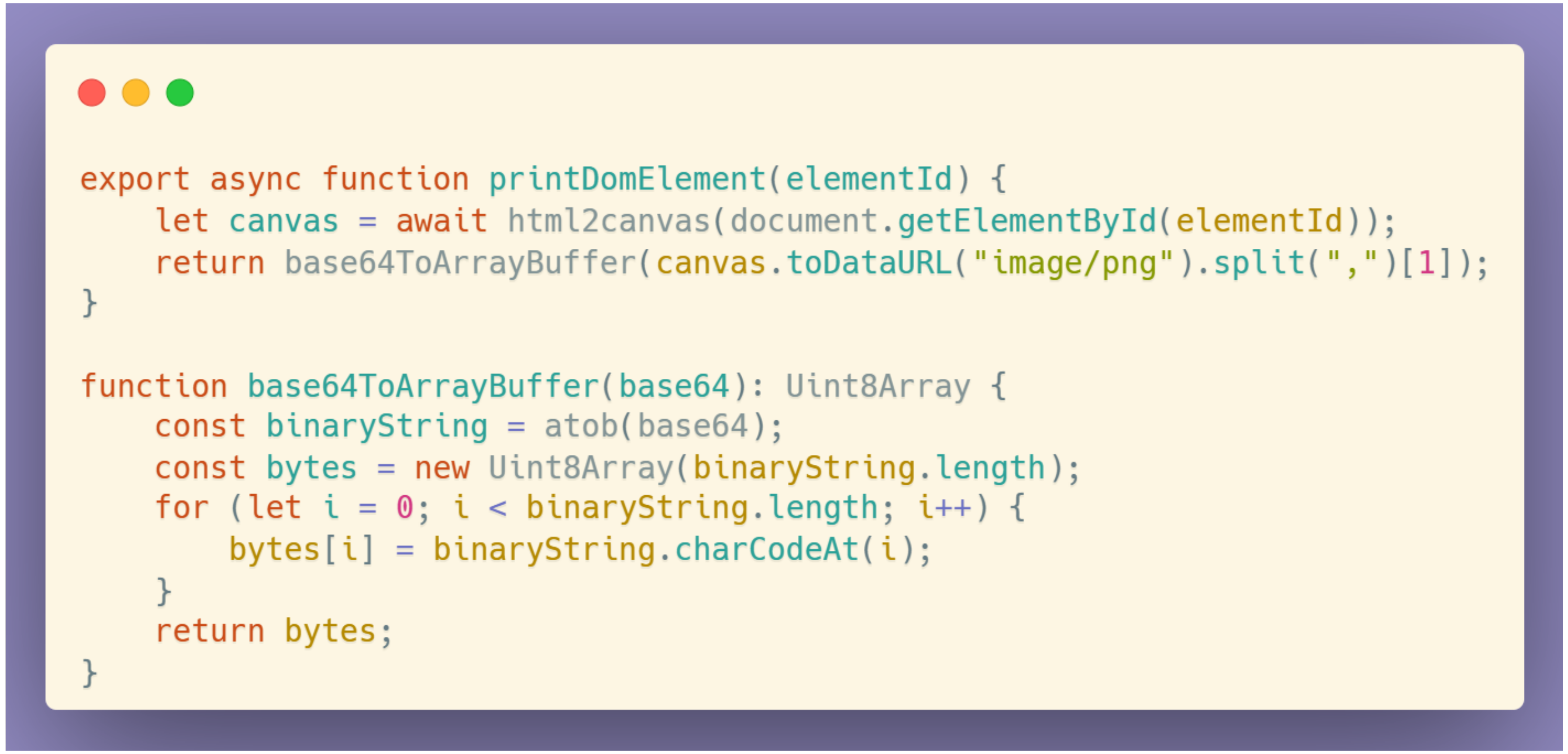
```
<script>
    window.getWindowWidth = () => {
        return window.innerWidth;
    };
</script>
```



```
[Inject]
public required IJSRuntime JSRuntime { get; set; }

protected async Task OnAfterRenderAsync(bool firstRender)
{
    double width = await JSRuntime.InvokeAsync<double>("getWindowWidth");
}
```

Escaping Back into JavaScript…

module.js

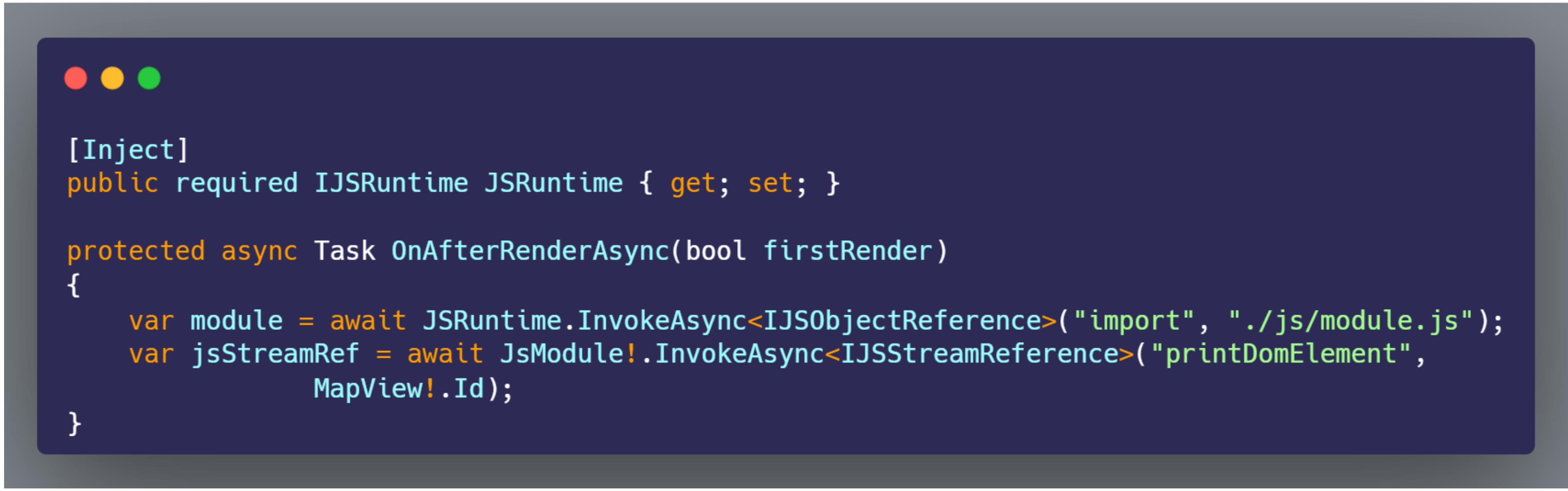


```
export async function printDomElement(elementId) {
    let canvas = await html2canvas(document.getElementById(elementId));
    return base64ToArrayBuffer(canvas.toDataURL("image/png").split(",")[1]);
}

function base64ToArrayBuffer(base64): Uint8Array {
    const binaryString = atob(base64);
    const bytes = new Uint8Array(binaryString.length);
    for (let i = 0; i < binaryString.length; i++) {
        bytes[i] = binaryString.charCodeAt(i);
    }
    return bytes;
}
```

techbash

Escaping Back into JavaScript...

Importing a JavaScript module



```
[Inject]
public required IJSRuntime JSRuntime { get; set; }

protected async Task OnAfterRenderAsync(bool firstRender)
{
    var module = await JSRuntime.InvokeAsync<IJSObjectReference>("import", "./js/module.js");
    var jsStreamRef = await JsModule!.InvokeAsync<IJSStreamReference>("printDomElement",
                MapView!.Id);
}
```

techbash

Escaping Back into JavaScript...

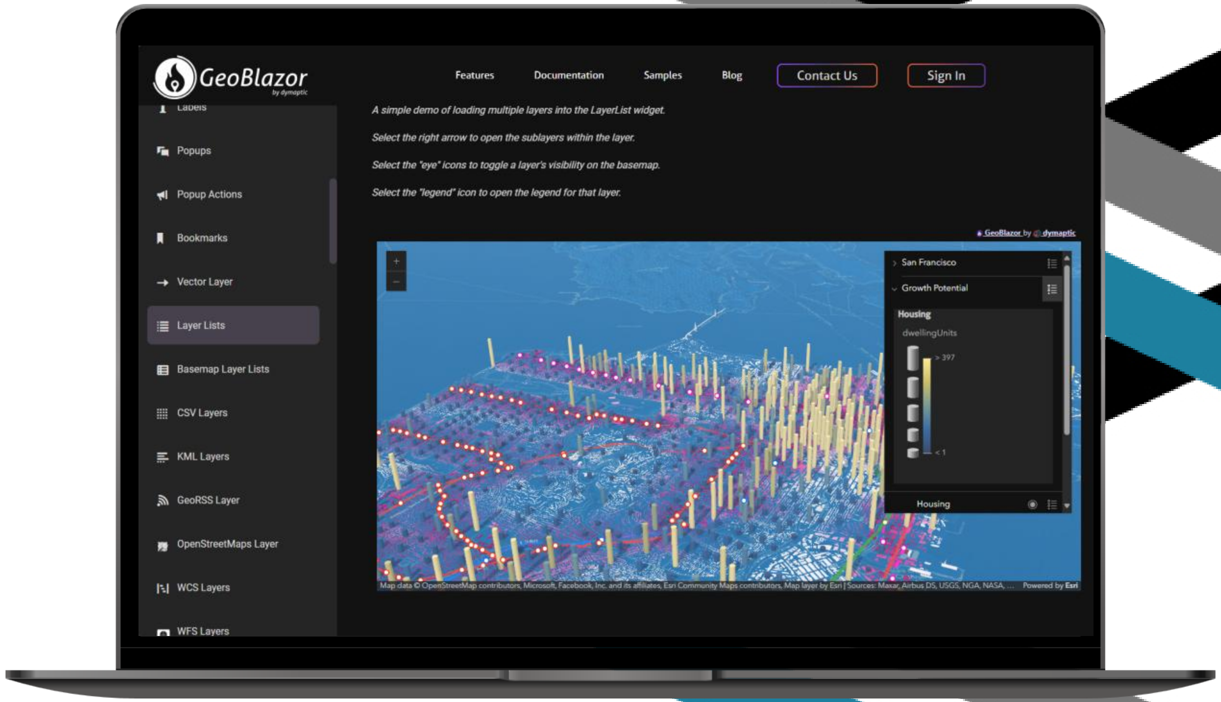
Calling .NET from JavaScript

```javascript
window.initialize = (dotNetRef) => {
  window.addEventListener('resize', async () => {
      const width = window.innerWidth;
      const height = window.innerHeight;
      await dotNetRef?.invokeMethodAsync('OnViewSizeChanged', width, height);
  });
}
```

```csharp
[JSInvokable]
public async Task OnViewSizeChanged(double width, double height)
{
    // update C# code
}
```

Check out https://samples.geoblazor.com

- Fully interactive application samples written in C# and Razor
- Each page is written to run in both Client and Server mode (live sample is Client mode)
- GeoBlazor library utilizes JSRuntime to interact with the ArcGIS Maps SDK for JavaScript, so GeoBlazor *users* don't have to switch to JavaScript



A simple demo of loading multiple layers into the LayerList widget.

Select the right arrow to open the sublayers within the layer.

Select the "eye" icons to toggle a layer's visibility on the basemap.

Select the "legend" icon to open the legend for that layer.

Map data © OpenStreetMap contributors, Microsoft, Facebook, Inc. and its affiliates, Esri Community Maps contributors, Map layer by Esri | Sources: Maxar, Airbus DS, USGS, NGA, NASA, ... Powered by Esri

Thank You!

dymaptic

Notes & Links @
https://timpurdum.dev

techbash

GeoBlazor