Blazor

Wael Kdouh - @waelkdouh

Senior Customer Engineer

v1.0

Module 2-1: Blazor

Module Overview

Why Do We Need a .Net Single Page App (SPA) Framework?

- The evolution of JavaScript in the recent years resulted in an ever more complex build process and byzantine build systems
- With JavaScript it seems the simpler the programming model gets, the more complex the build system and tooling becomes
- This isn't just about 'language' either. JavaScript's insane build systems required for all major frameworks these days is a house of cards that seems to break anytime you step away for more than a few days. Other platforms have skinned that cat in other and potentially more efficient ways that are easier and more integrated without the brittleness that seems to come part and parcel for JavaScript development

“Rick Strahl“

Module 2-1: Blazor

Section 1: WebAssembly

Lesson: WebAssembly Fundamentals

What Is Webassembly?

WebAssembly is a new type of code that can be run in modern web browsers — it is a low-level assembly-like language with a compact binary format that runs with near-native performance and provides languages such as C/C++ and Rust with a compilation target so that they can run on the web. It is also designed to run alongside JavaScript, allowing both to work together

“MDN”

What Is Webassembly?

A Virtual Machine For The Web

Again.

How Is It Different Than Java?

Separate VM

Clunky

No Dom Integration

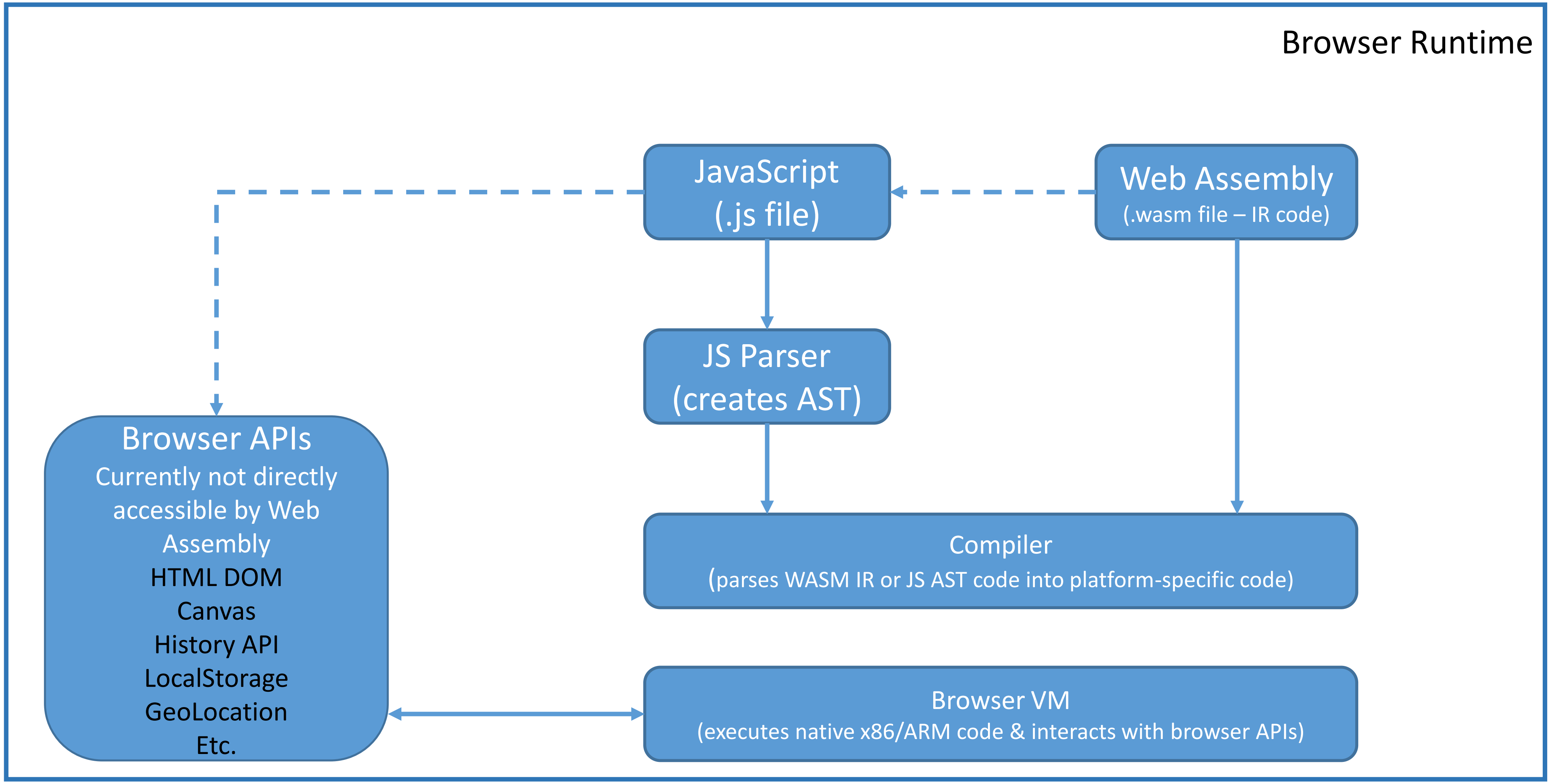
What Is Different This Time?

Exposes Internal Browser VM

Cross-Browser Support

JS Bindings

Re-Assembling The Web



Browser Runtime

JavaScript
(.js file)

Web Assembly
(.wasm file – IR code)

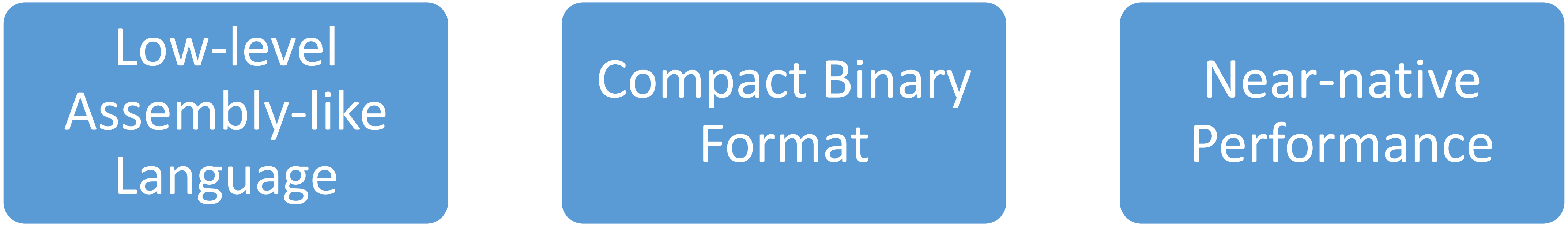
JS Parser
(creates AST)

Browser APIs
Currently not directly accessible by Web Assembly
HTML DOM
Canvas
History API
LocalStorage
GeoLocation
Etc.

Compiler
(parses WASM IR or JS AST code into platform-specific code)

Browser VM
(executes native x86/ARM code & interacts with browser APIs)

Let’s Break That Down…



Low-level Assembly-like Language

Compact Binary Format

Near-native Performance

Web Assembly Goals

Fast, Efficient, Portable

Readable, Debuggable

Keep Secure

Don’t Break The Web

Module 2-1: Blazor

Section 2: Blazor

Lesson: Getting Started

WebAssembly and Mono

- Mono is an open source implementation of the .NET CLI specification, meaning that Mono is a platform for running .NET assemblies

- Mono is written in C++. This is important because you can compile C++ to WebAssembly

- The Mono team decided to try to compile Mono to WebAssembly, which they did successfully which in turn gave the birth to Blazor

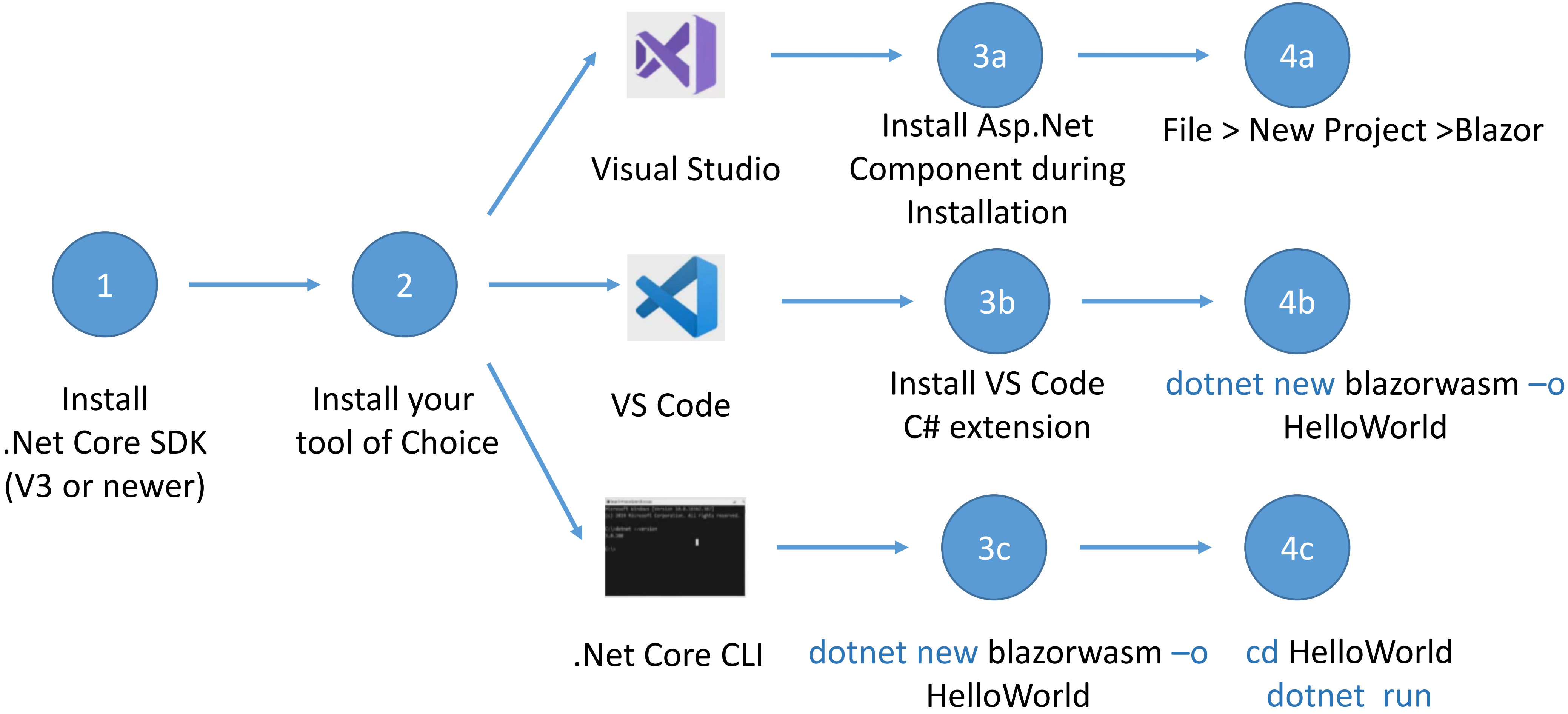
Approaches For Compiling Mono To WebAssembly

- .Net team considered two approaches:
  - Take the .NET code and compile it together with the Mono runtime into one big WASM application. This approach takes a lot of time because you need to take several steps to compile everything into WASM, which is not so practical for day-to-day development
  - The second approach takes the Mono runtime, compiles it into WASM, and this runs in the browser where it will execute .NET Intermediate Language just like normal .NET does. This has a big advantage as you can simply run .NET assemblies without having to compile them first into WASM. **This is the approach currently taken by Blazor**. The **disadvantage of this is that it needs to download a lot of .NET assemblies**. This **can be solved by using Tree Shaking algorithms**, which removes all unused code from assemblies

Blazor History

- NDC Demo In 2017
- Lots Of Interest…
- Moved To ASP.Net’s Github As Official Experiment Jan 2018
- Server Side Blazor Components Shipped With .NET 3.0
- Client Side Blazor Shipping After .NET 3.0

Blazor Development Environment Options



1

Install
.Net Core SDK
(V3 or newer)

2

Install your
tool of Choice

Visual Studio

3a

Install Asp.Net
Component during
Installation

4a

File > New Project >Blazor

VS Code

3b

Install VS Code
C# extension

4b

dotnet new blazorwasm –o
HelloWorld

.Net Core CLI

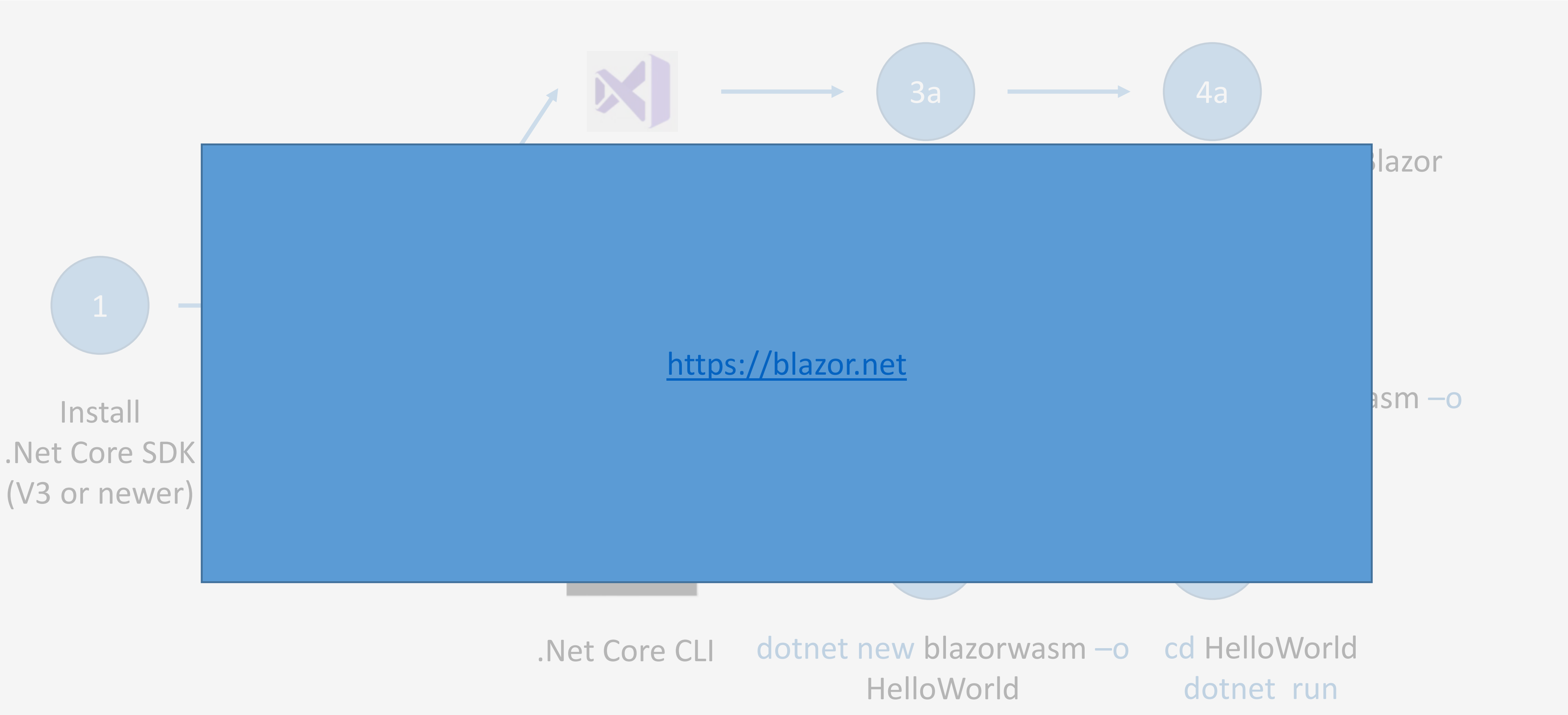
3c

dotnet new blazorwasm –o
HelloWorld

4c

cd HelloWorld
dotnet run

Blazor Development Environment Options



Install
.Net Core SDK
(V3 or newer)

https://blazor.net

.Net Core CLI

dotnet new blazorwasm –o HelloWorld

cd HelloWorld
dotnet run

Why Use Blazor?

WebAssembly is supported by all major browsers

Use C# For Interactive Web Applications

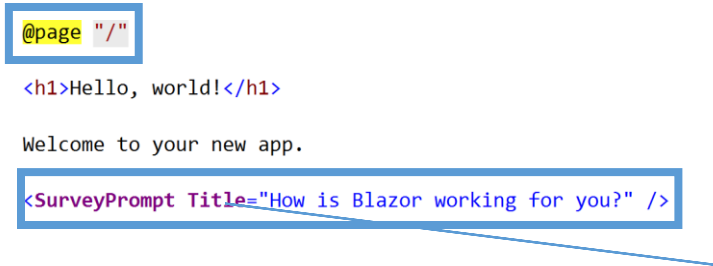
Reuse Existing Libraries

Performance Is Near Native

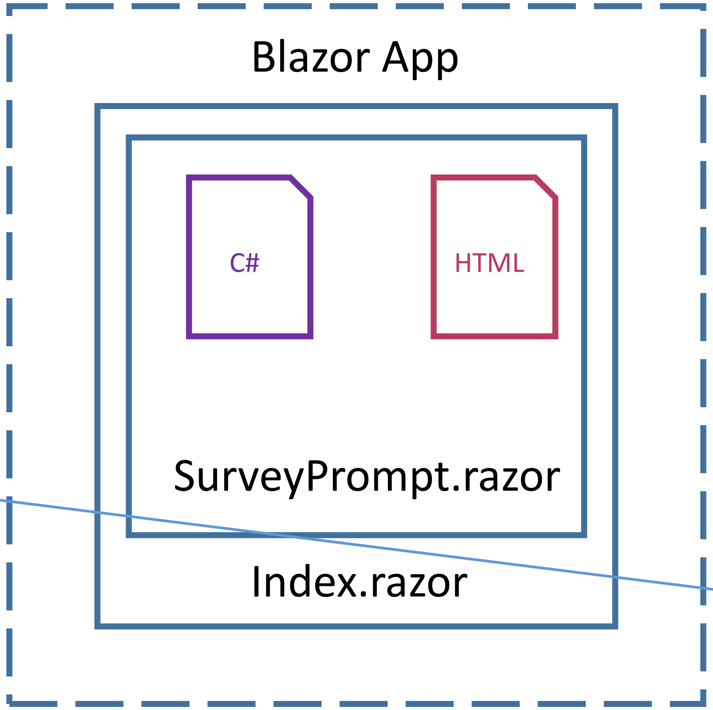
Tooling And Debugging

What Is Blazor?

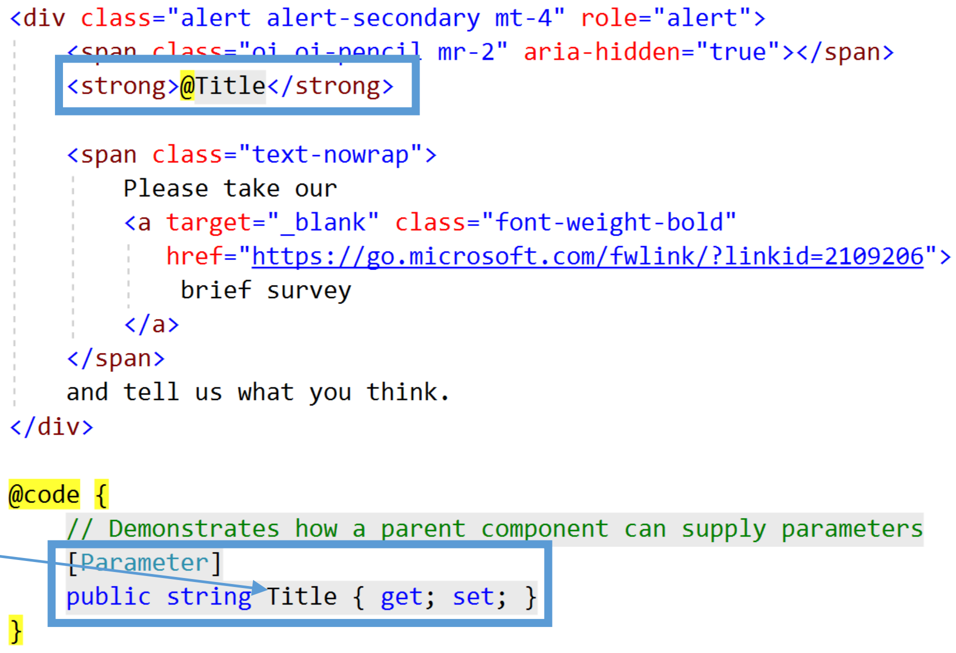
Index.razor



```
@page "/"

<h1>Hello, world!</h1>

Welcome to your new app.

<SurveyPrompt Title="How is Blazor working for you?" />
```



SurveyPrompt.razor



```
<div class="alert alert-secondary mt-4" role="alert">
    <span class="oi oi-pencil mr-2" aria-hidden="true"></span>
    <strong>@Title</strong>

    <span class="text-nowrap">
        Please take our
        <a target="_blank" class="font-weight-bold"
           href="https://go.microsoft.com/fwlink/?linkid=2109206">
            brief survey
        </a>
    </span>
    and tell us what you think.
</div>

@code {
    // Demonstrates how a parent component can supply parameters
    [Parameter]
    public string Title { get; set; }
}
```

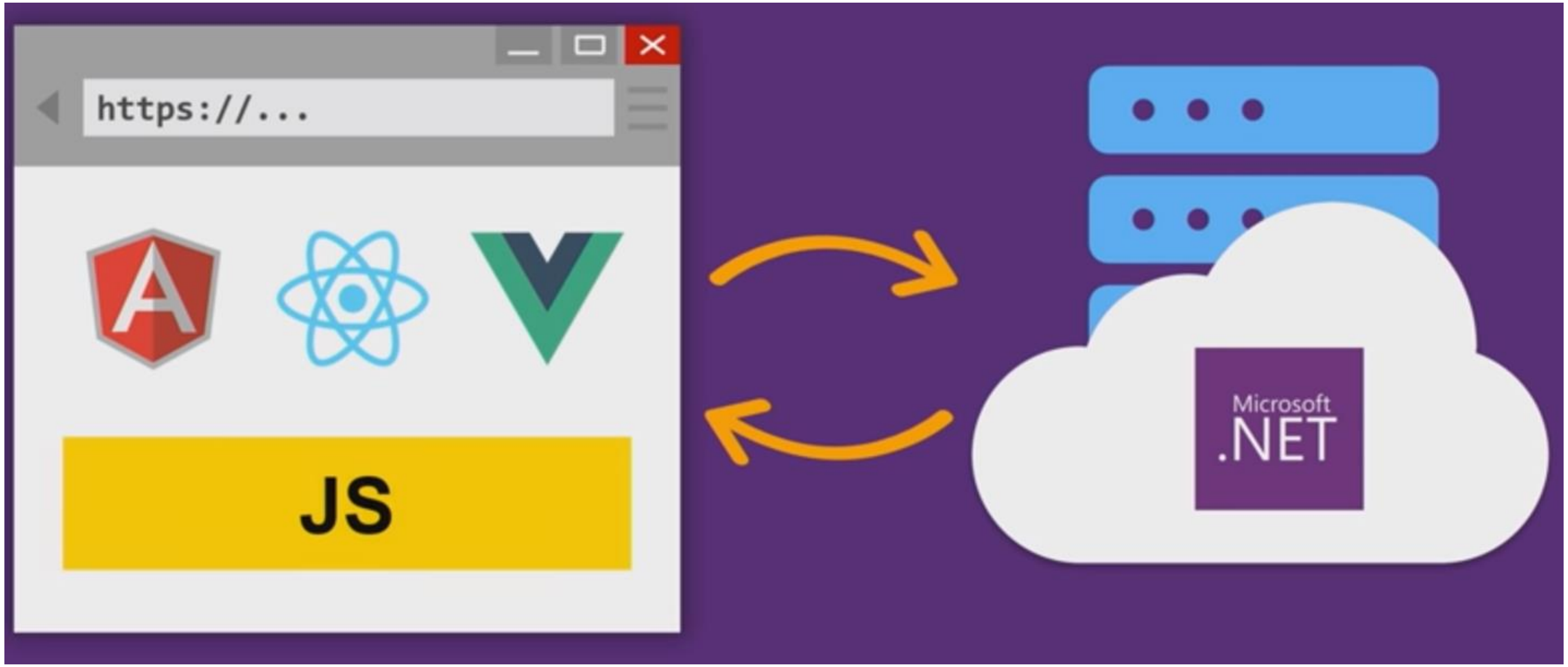
Demo: A Quick Hello World

Module 2-1: Blazor

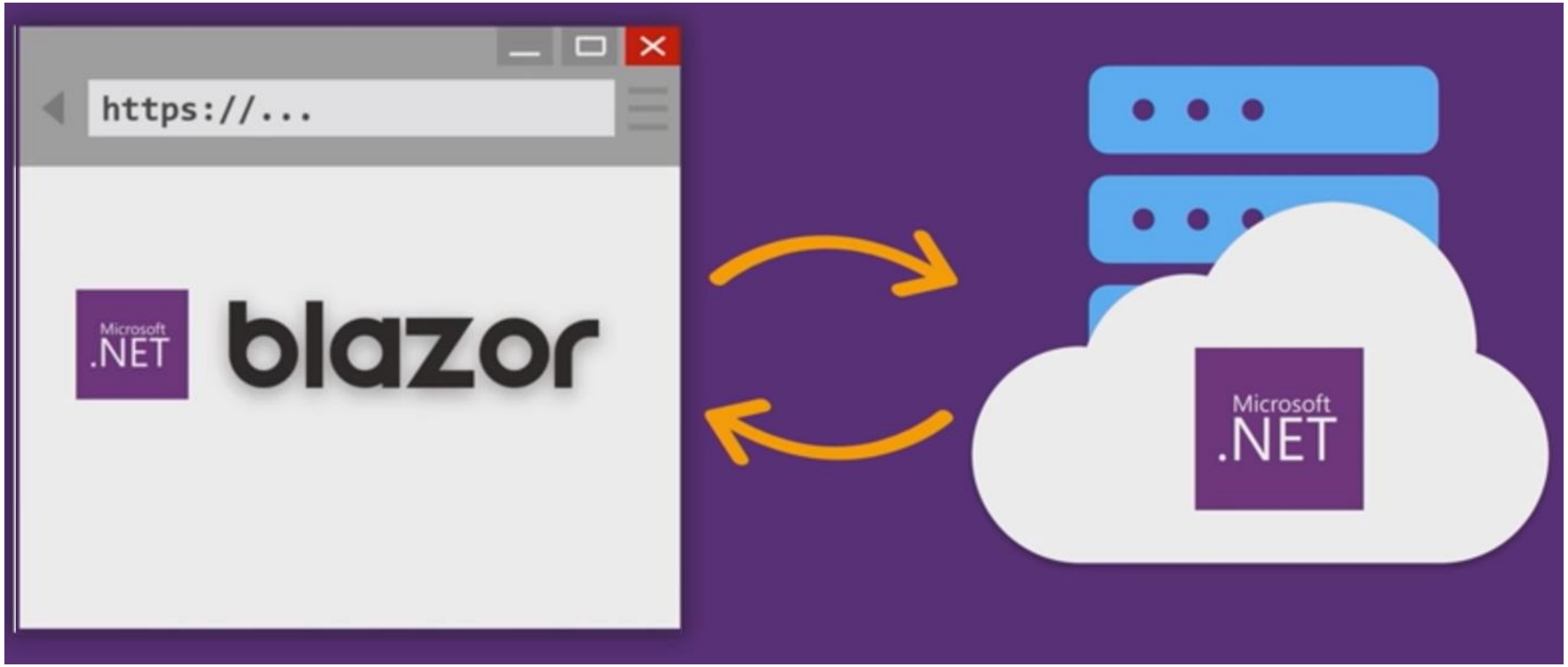
Section 2: Blazor

Lesson: Hosting Models

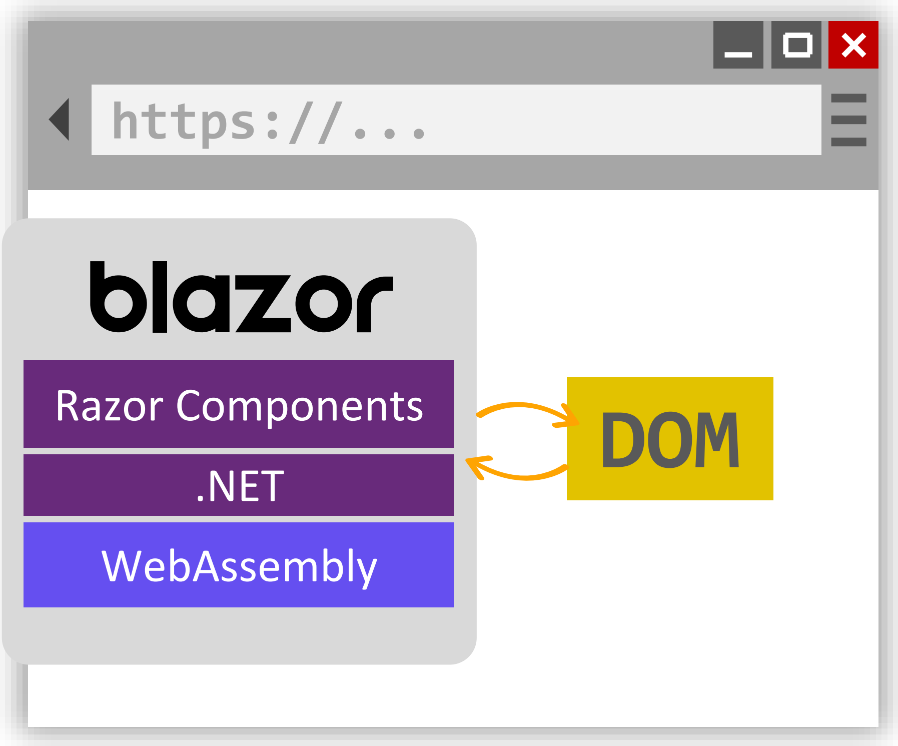
Blazor WebAssembly



Blazor WebAssembly



Blazor WebAssembly



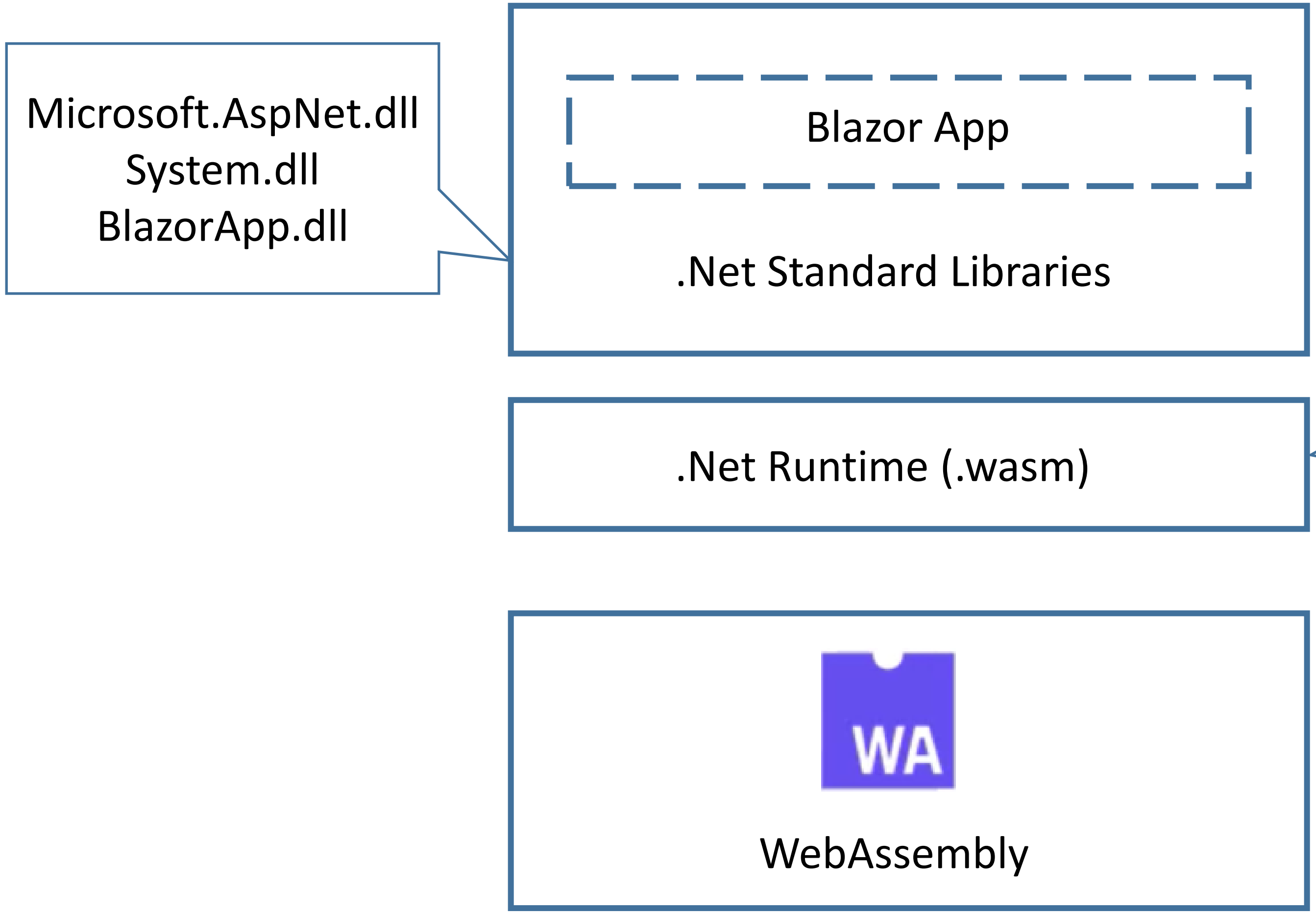
Browser provides the core WebAssembly support

.Net WebAssembly runtime runs on top of WebAssembly

Razor Components run in the browser on top of the .Net runtime

Blazor receives UI events from the DOM and later sends back DOM diffs to be applied to the DOM

Blazor WebAssembly



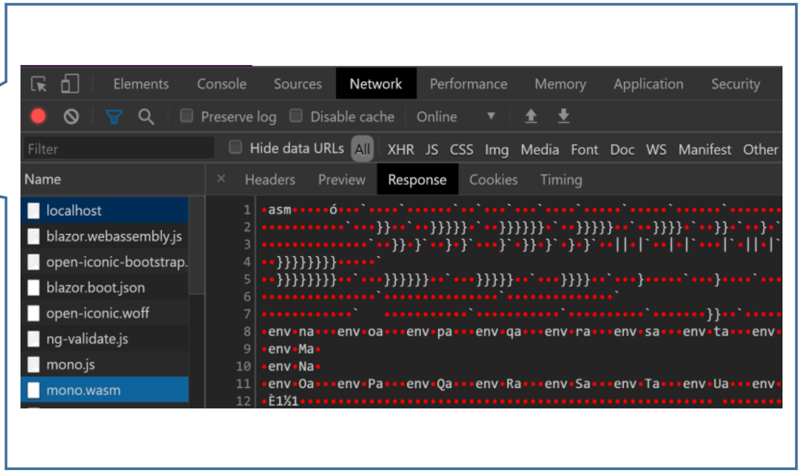
Microsoft.AspNet.dll
System.dll
BlazorApp.dll

Blazor App

.Net Standard Libraries

.Net Runtime (.wasm)

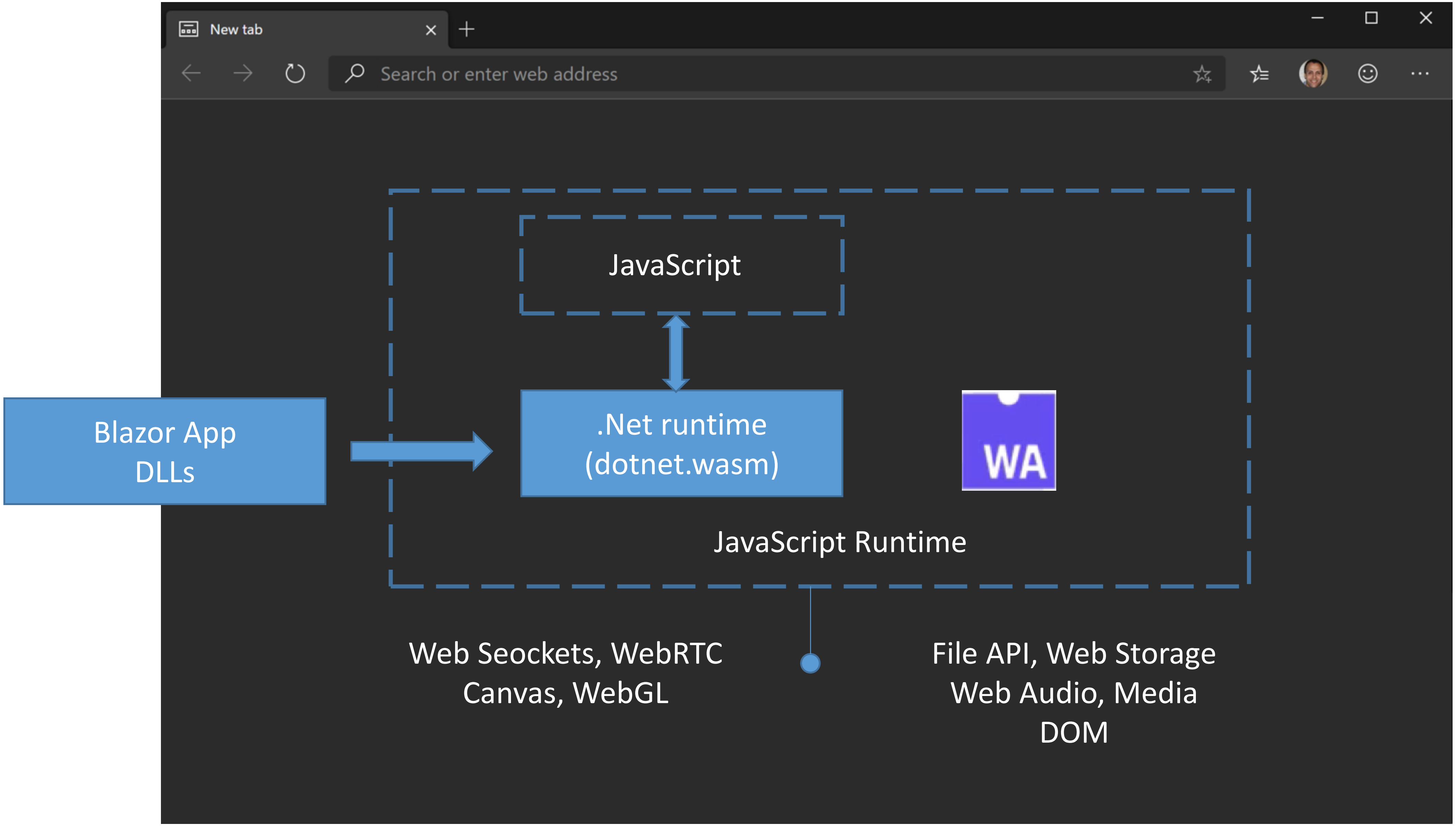
WebAssembly



localhost
blazor.webassembly.js
open-iconic-bootstrap.
blazor.boot.json
open-iconic.woff
ng-validate.js
mono.js
mono.wasm

```
•asm•••••ó
•env•na•••env•oa•••env•pa•••env•qa•••env•ra•••env•sa•••env•ta•••env•
•env•Ma•
•env•Na•
•env•Oa•••env•Pa•••env•Qa•••env•Ra•••env•Sa•••env•Ta•••env•Ua•••env•
•È1½1
```

Blazor WebAssembly SandBox



JavaScript

Blazor App
DLLs

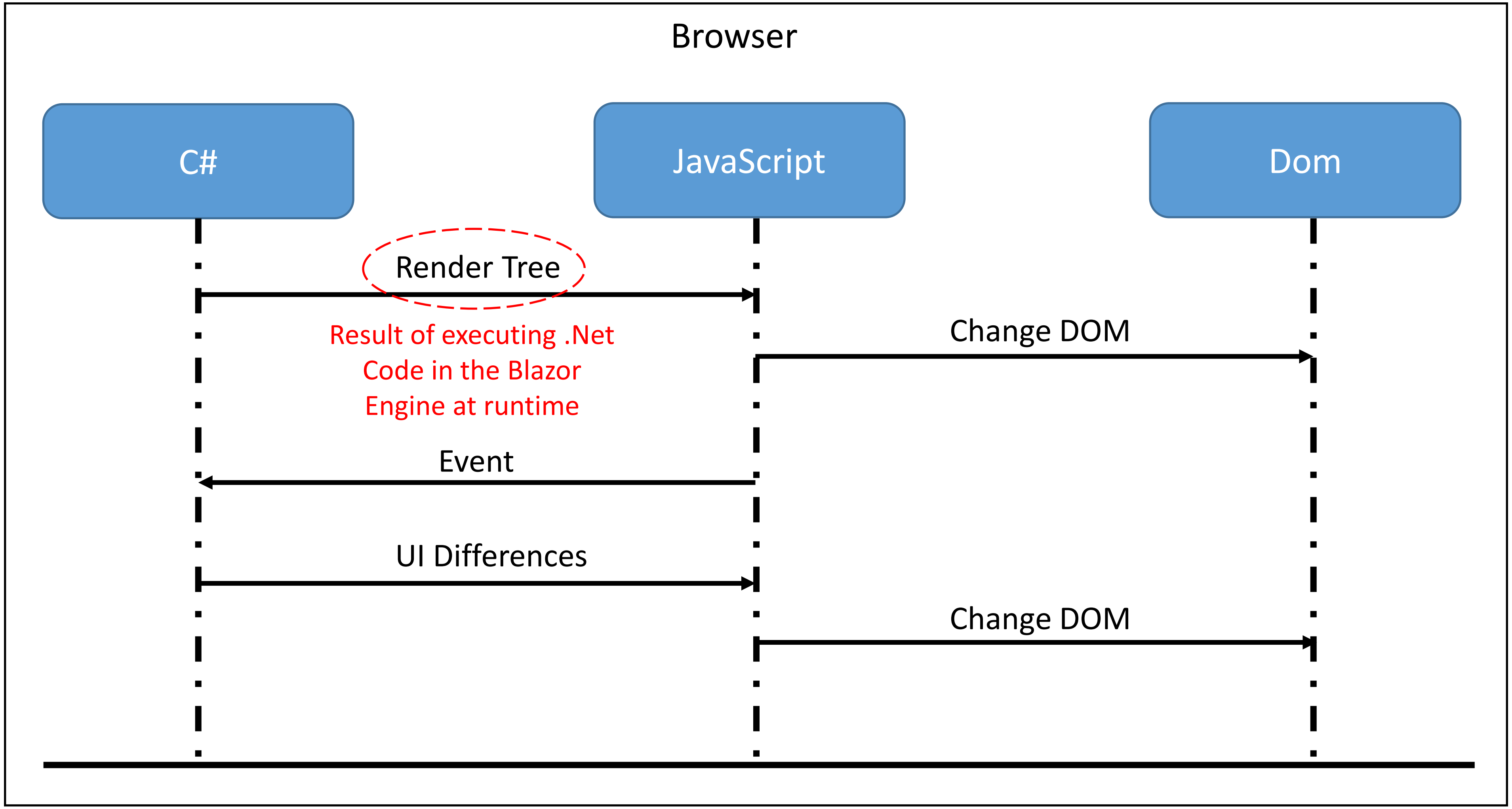
.Net runtime
(dotnet.wasm)

JavaScript Runtime

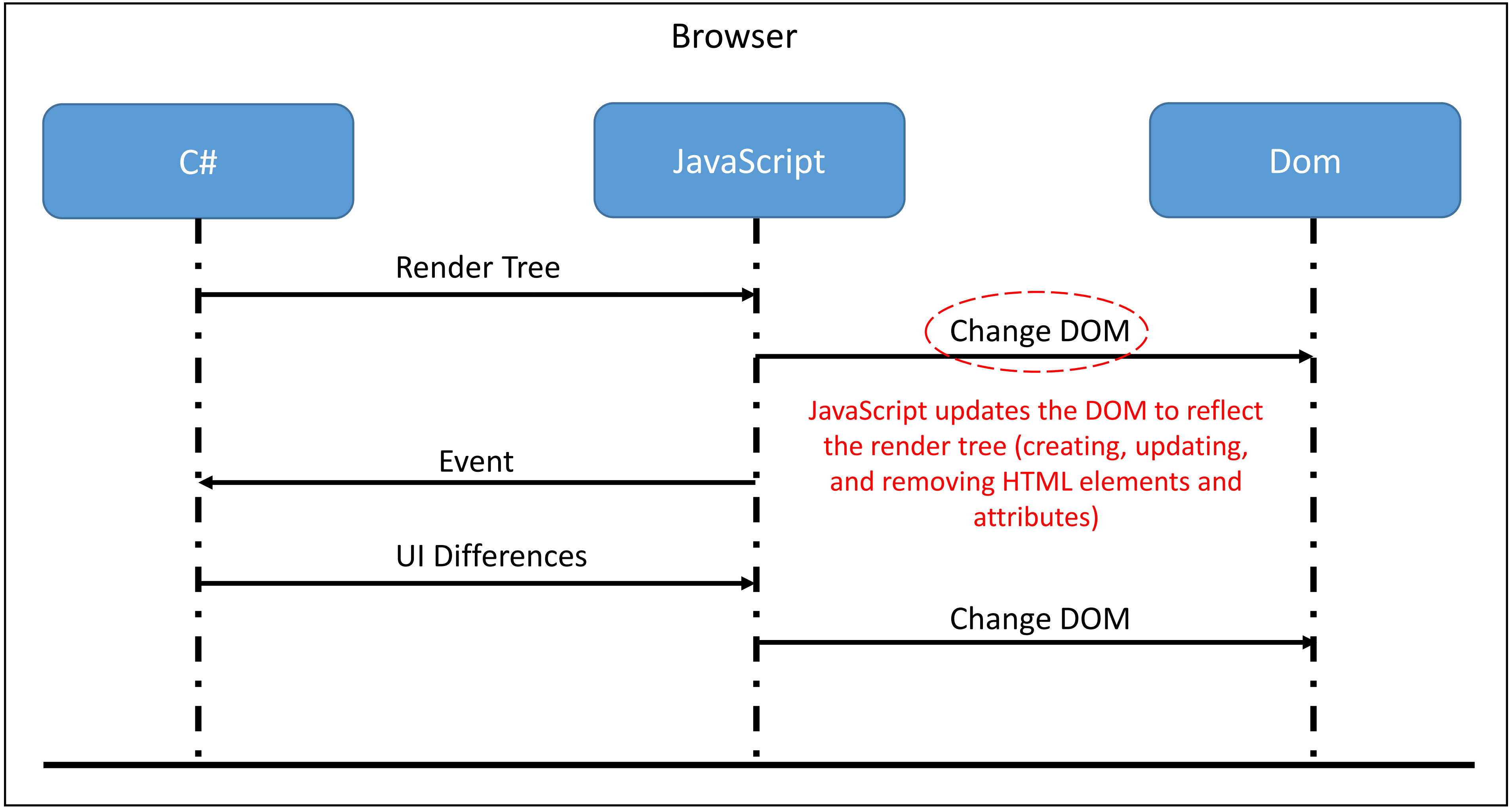
Web Seockets, WebRTC
Canvas, WebGL

File API, Web Storage
Web Audio, Media
DOM

Blazor WebAssembly DOM Generation Process

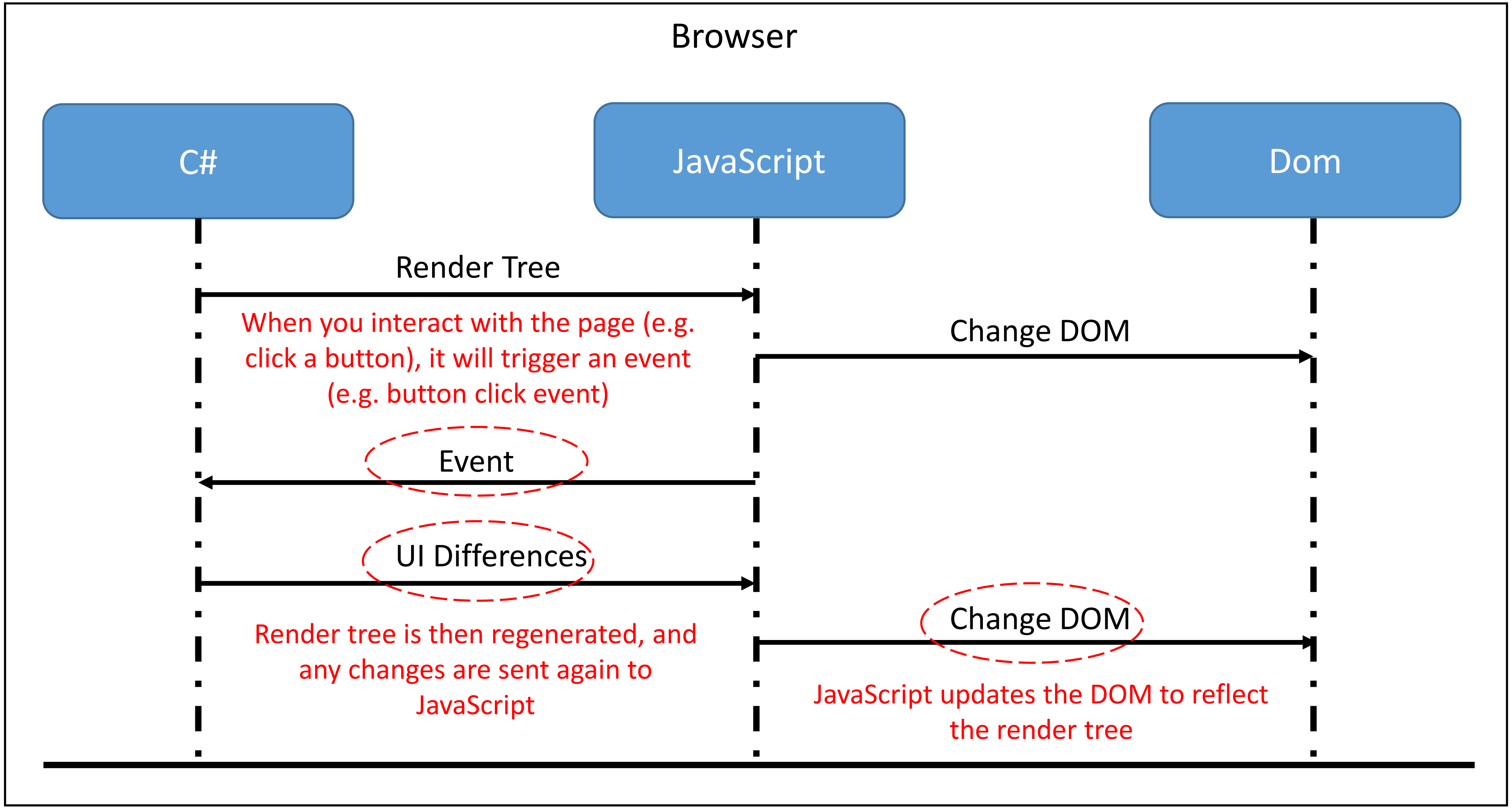


Blazor WebAssembly DOM Generation Process



JavaScript updates the DOM to reflect the render tree (creating, updating, and removing HTML elements and attributes)

Blazor WebAssembly DOM Generation Process



Browser

C#

JavaScript

Dom

Render Tree

When you interact with the page (e.g. click a button), it will trigger an event (e.g. button click event)

Change DOM

Event

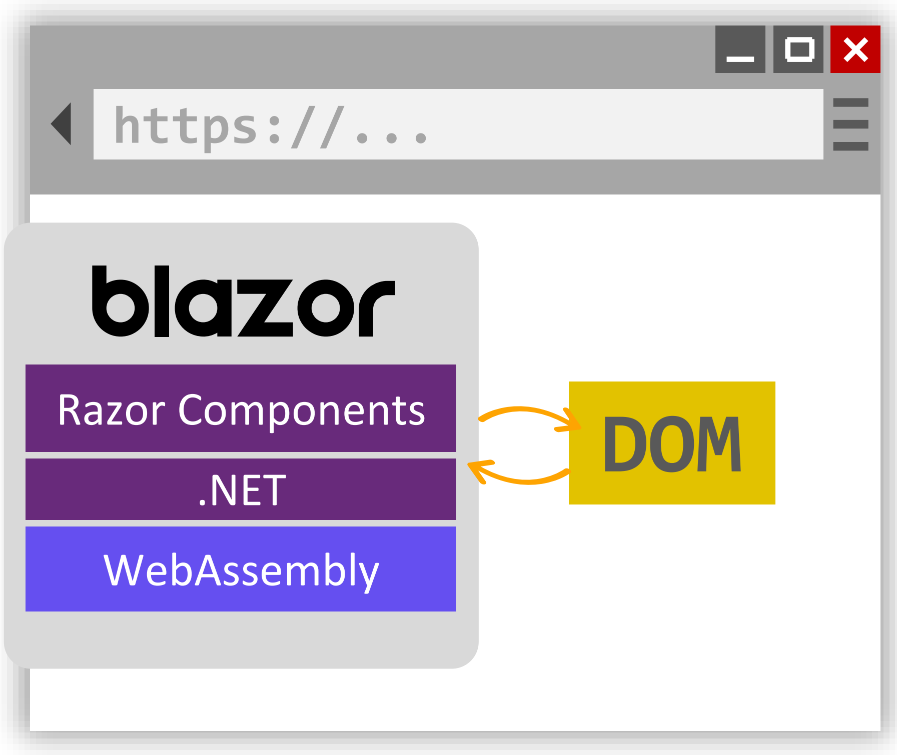
UI Differences

Render tree is then regenerated, and any changes are sent again to JavaScript

Change DOM

JavaScript updates the DOM to reflect the render tree

Hosting Blazor WebAssembly

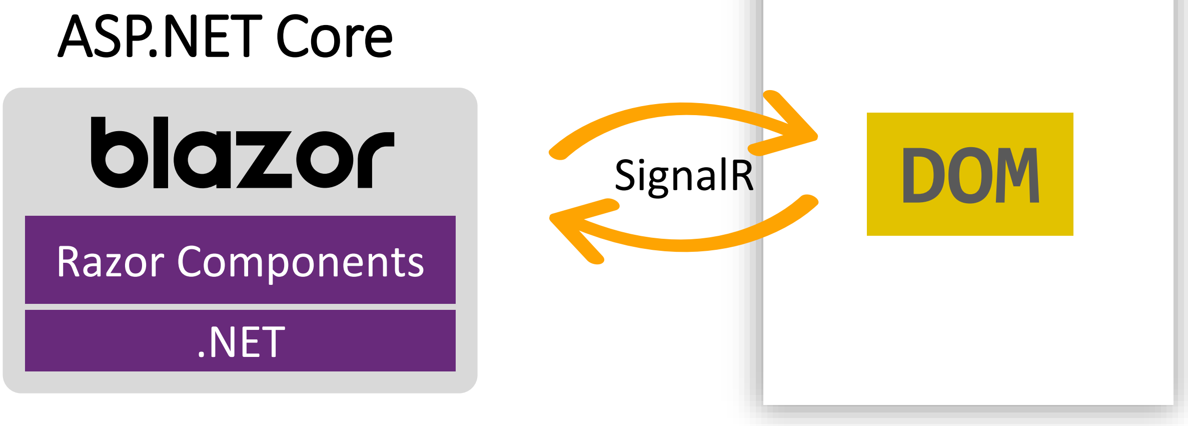


Azure
CDN

Azure
App
Service

Azure Storage
Static Website

Server-Side Blazor – Released With .Net Core 3

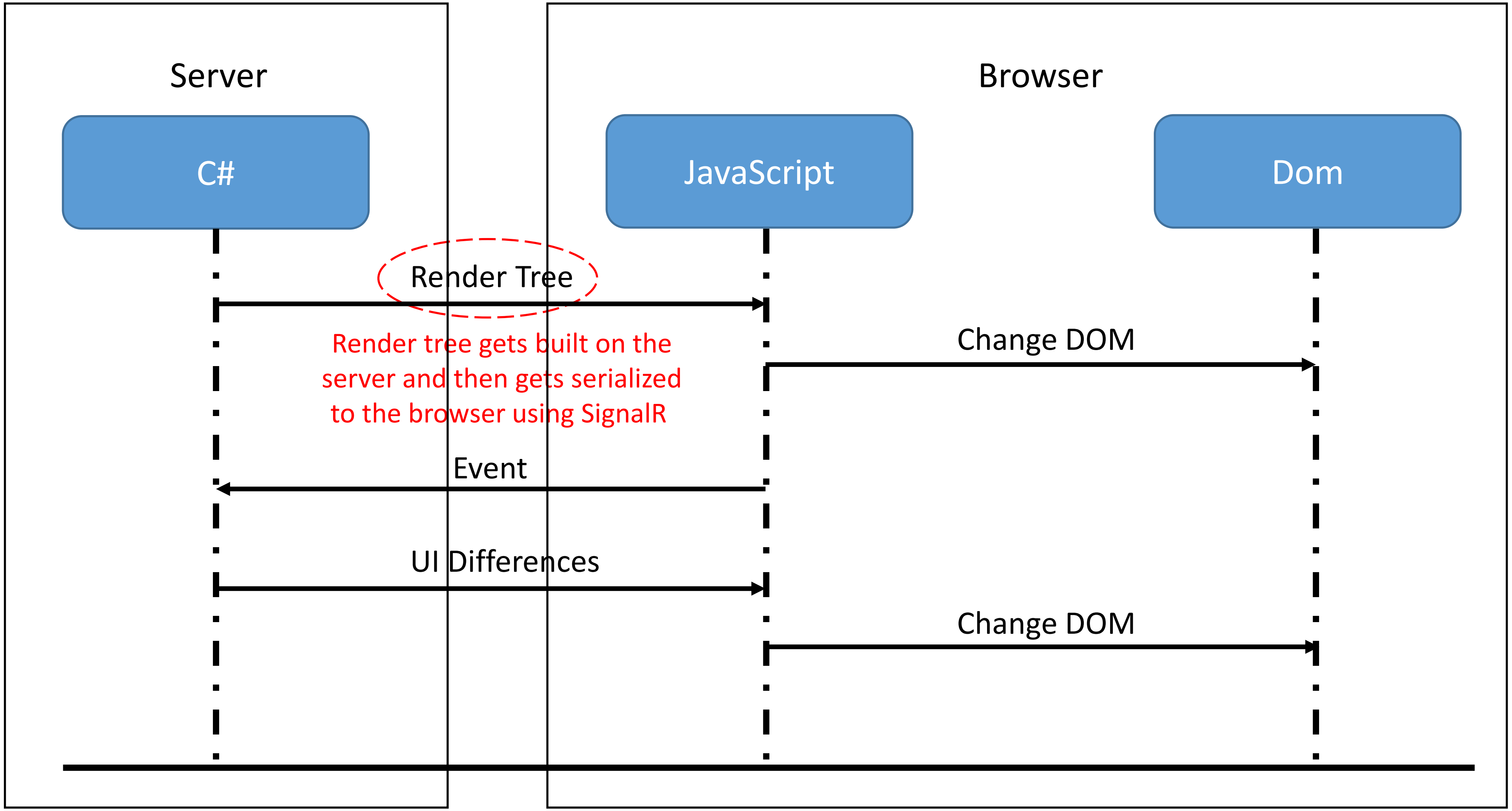


App is executed on the server from within an ASP.NET Core app

Browser Sets Up Realtime SignalR connection to handle and send all the UI events that the user creates

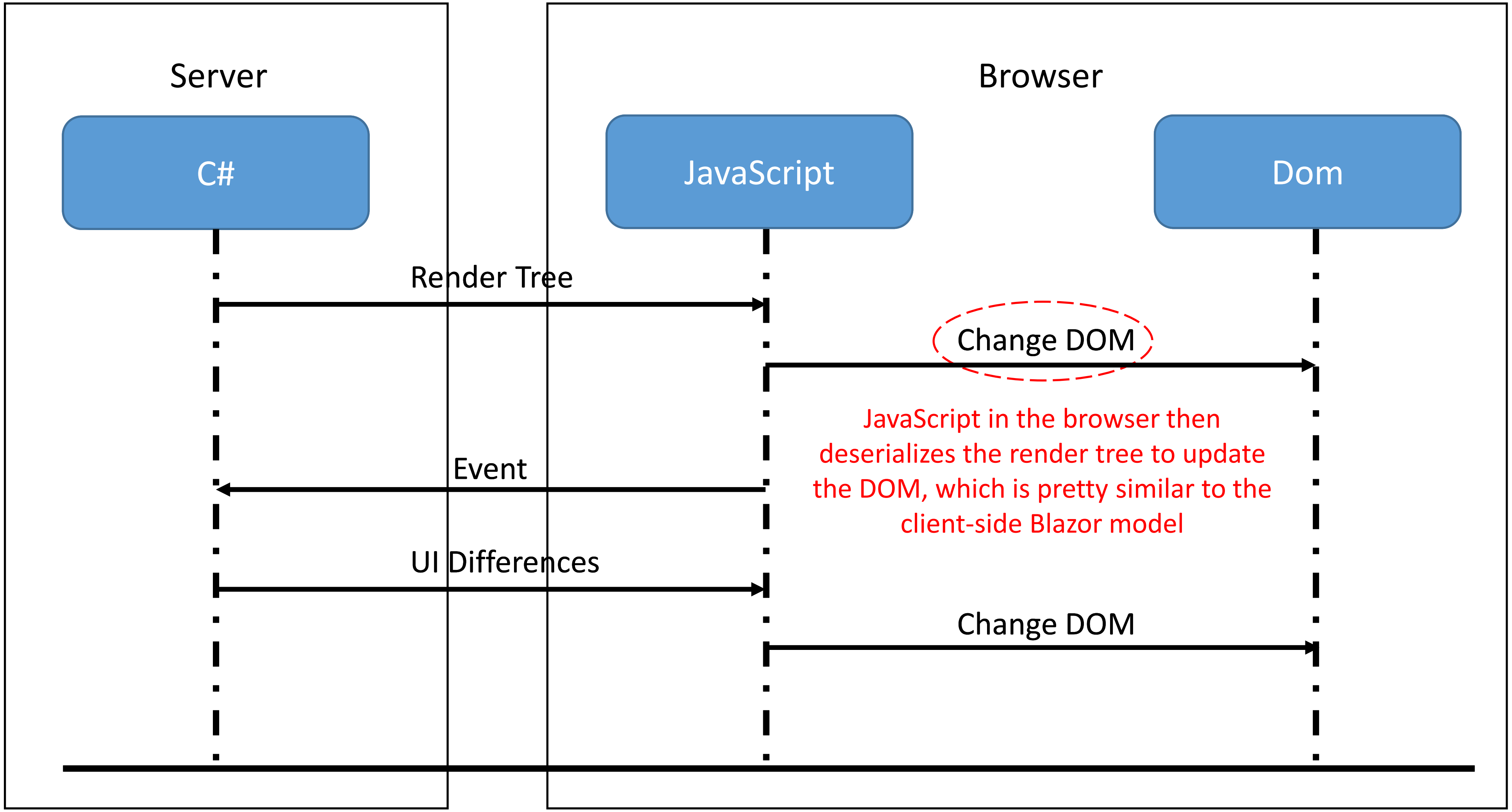
Blazor receives UI events from the DOM and later sends back DOM diffs using the SignalR connection to be applied to the Dom

Server-Side Blazor DOM Generation Process



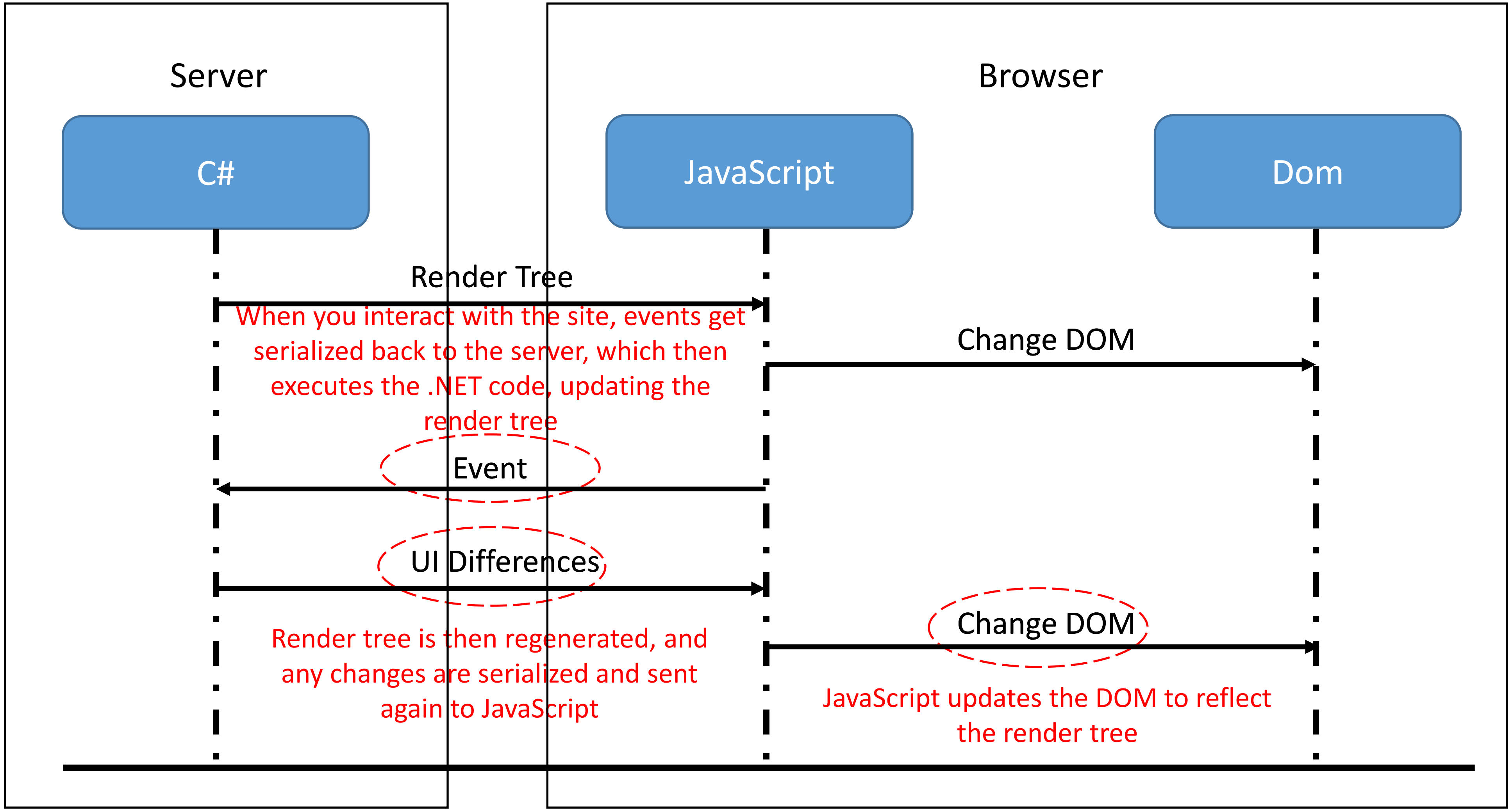
Render tree gets built on the server and then gets serialized to the browser using SignalR

Server-Side Blazor DOM Generation Process



JavaScript in the browser then deserializes the render tree to update the DOM, which is pretty similar to the client-side Blazor model

Server-Side Blazor DOM Generation Process

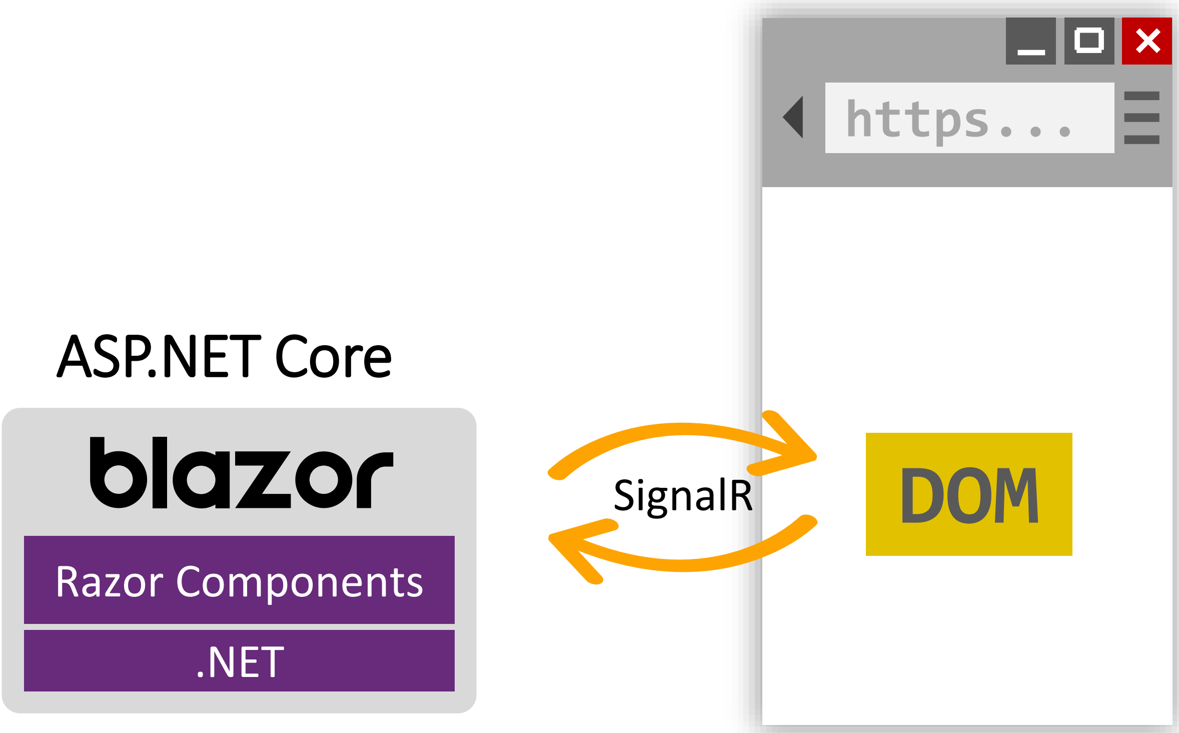


When you interact with the site, events get serialized back to the server, which then executes the .NET code, updating the render tree

Render tree is then regenerated, and any changes are serialized and sent again to JavaScript

JavaScript updates the DOM to reflect the render tree

Hosting Server-Side Blazor



ASP.NET Core

blazor

Razor Components

.NET

SignalR

https...

DOM

Azure SignalR
Service

Azure
App
Service

Blazor WebAssembly Vs Blazor Server

Blazor WebAssembly

- Pros:
  - True SPA, full interactivity
  - Near Native Performance
  - Utilize client resources
  - No server needed
  - Can work offline
- Cons:
  - Restricted to the capabilities of the browser
  - Larger download size
  - Requires WebAssembly
  - Client-side secrets

Blazor Server

- Pros:
  - Smaller download size, faster load time
  - Running on fully featured .NET runtime
  - Code never leaves the server, Server side secrets
- Cons:
  - Latency
  - No offline support
  - Consumes more server resources

Which Hosting Model To Use When?

	Blazor WebAssembly	Blazor Server
Need near native experience	●	
Need to connect to Server-Side resources		●
Using Older Browsers		●
Offline Support	●	
You don’t want to run an Asp.Net Core Server	●	
You want to create a fast, interactive web app with C#	●	●

Demo: Hosting Models

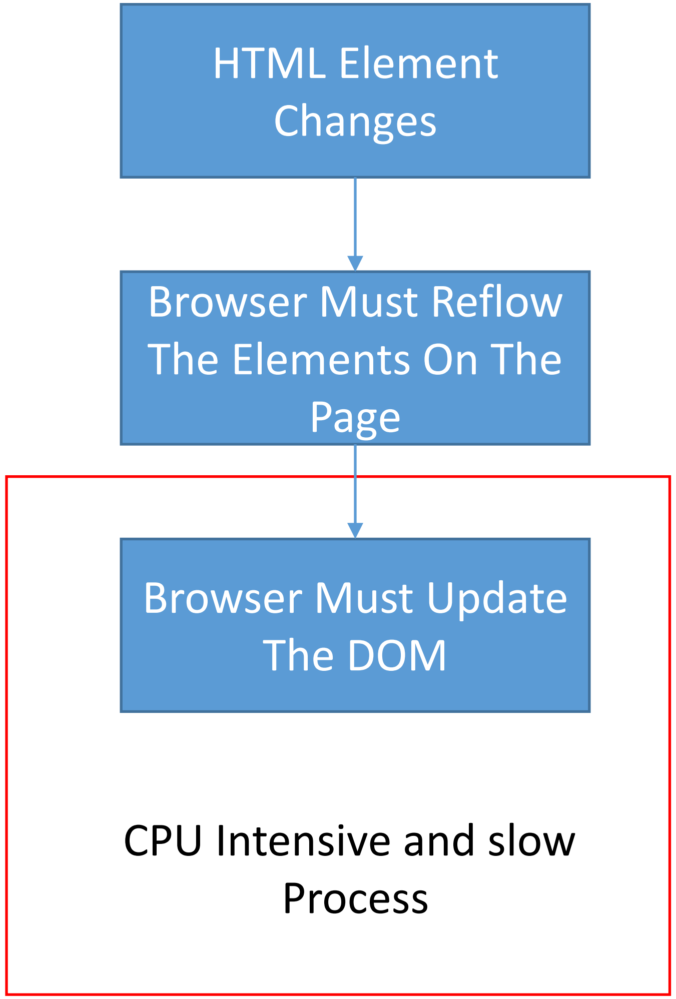
Manipulating DOM Elements From Blazor

- Blazor is specifically designed to provide a convenient and efficient way to update the UI without having to issue individual calls to mutate individual DOM elements

- **Blazor runs a diffing algorithm to work out the minimal set of changes to apply and runs them in a batch**. This will be far more performant than crossing the interop boundary separately for each element update, and potentially interleaving DOM mutations with repaints, etc. Most modern SPA frameworks are built around an architecture similar to this

- Blazor still does allow you to directly reach DOM elements if you want via its JS interop APIs, but you should only do that for exceptional custom things, not for routine UI rendering

Rendering Without Virtual DOM and Incremental DOM



HTML Element Changes

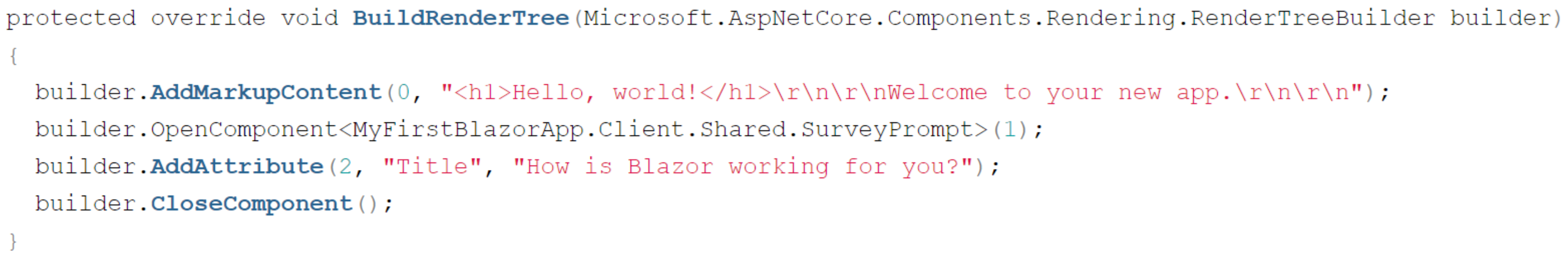
Browser Must Reflow The Elements On The Page

Browser Must Update The DOM

CPU Intensive and slow Process

Rendering With Virtual DOM and Incremental DOM

- **Virtual DOM**
  - An in-memory representation of the elements that will make up the HTML page. This data creates a tree of HTML elements as if they had been specified by an HTML mark-up page. Blazor components create this Virtual DOM in its Razor views via a virtual method named BuildRenderTree. For example, the BuildRenderTree for the standard Pages/Index.razor page looks like this



```
protected override void BuildRenderTree(Microsoft.AspNetCore.Components.Rendering.RenderTreeBuilder builder)
{
  builder.AddMarkupContent(0, "<h1>Hello, world!</h1>\r\n\r\nWelcome to your new app.\r\n\r\n");
  builder.OpenComponent<MyFirstBlazorApp.Client.Shared.SurveyPrompt>(1);
  builder.AddAttribute(2, "Title", "How is Blazor working for you?");
  builder.CloseComponent();
}
```

Rendering With Virtual DOM and Incremental DOM

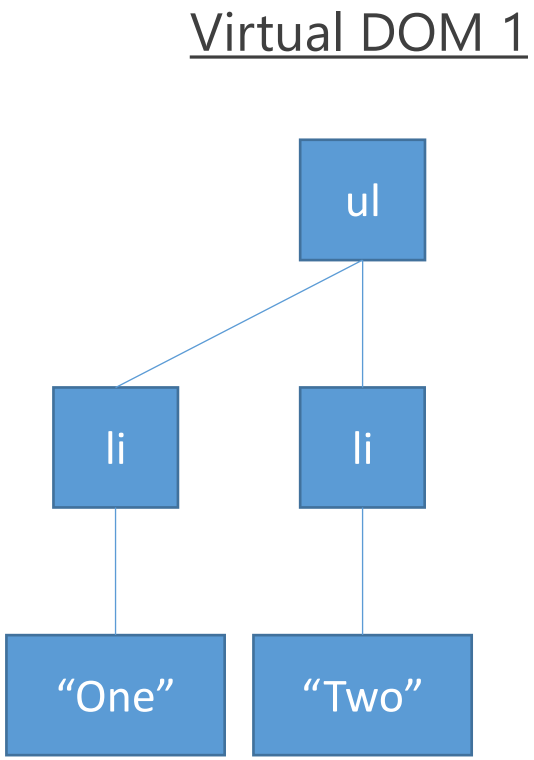
- **Virtual DOM**
  - Building a data tree that represents the view to be rendered has two significant benefits:
    - Attribute values of those virtual HTML elements can be updated many times in code during a complex update process without the browser having to re-render and reflow its view until after the process has finished
    - Render trees can be created by comparing two trees and building a new tree that is the difference between the two. **This allows us to utilize an Incremental DOM approach**

Rendering With Virtual DOM and Incremental DOM

- **Incremental DOM**
  - Incremental DOM is a technique that minimizes the amount of work needed to update the elements in a browser's view
  - **Being able to create a diff tree gives us the ability to represent changes to the view using the smallest number of changes possible required to update the DOM**. This saves time when changing the display (so the user-experience is better), and in Server-Side Blazor apps it means fewer bytes over the network – making a Blazor app more useable on slow networks or very remote locations

Rendering With Virtual DOM and Incremental DOM

Example – Adding a new list item

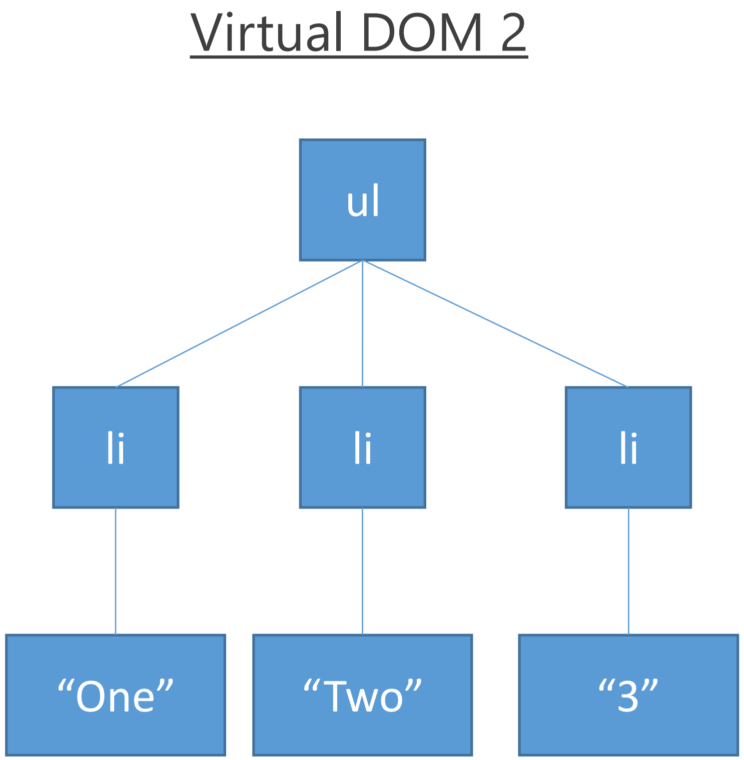


The current Virtual DOM for the view in the browser consists of a list with two items

Rendering With Virtual DOM and Incremental DOM

Example – Adding a new list item

The app adds a new item to the list. Blazor represents this in a new Virtual DOM

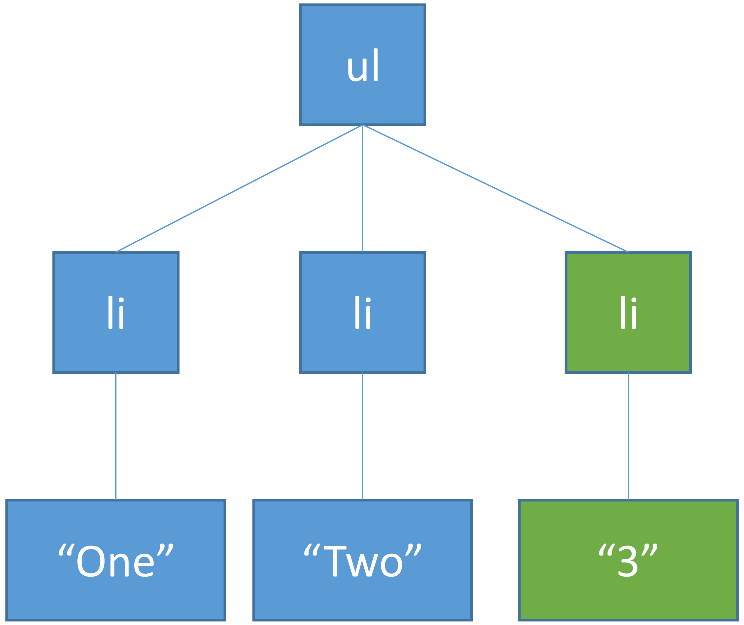


Rendering With Virtual DOM and Incremental DOM

Example – Adding a new list item

The following differential tree is determined to be the fewest number of changes required. In this case, one new <li> and one new text element "3"

Differential Tree

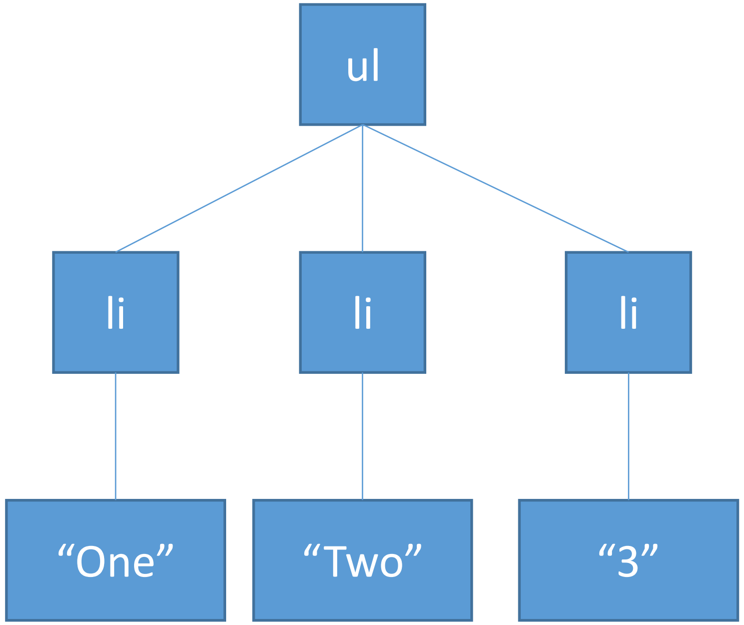


Rendering With Virtual DOM and Incremental DOM

Example – Adding a new list item

The differential render tree is then used to update the actual HTML DOM in the browser

Actual DOM



Demo: Rendering With Virtual DOM and Incremental DOM

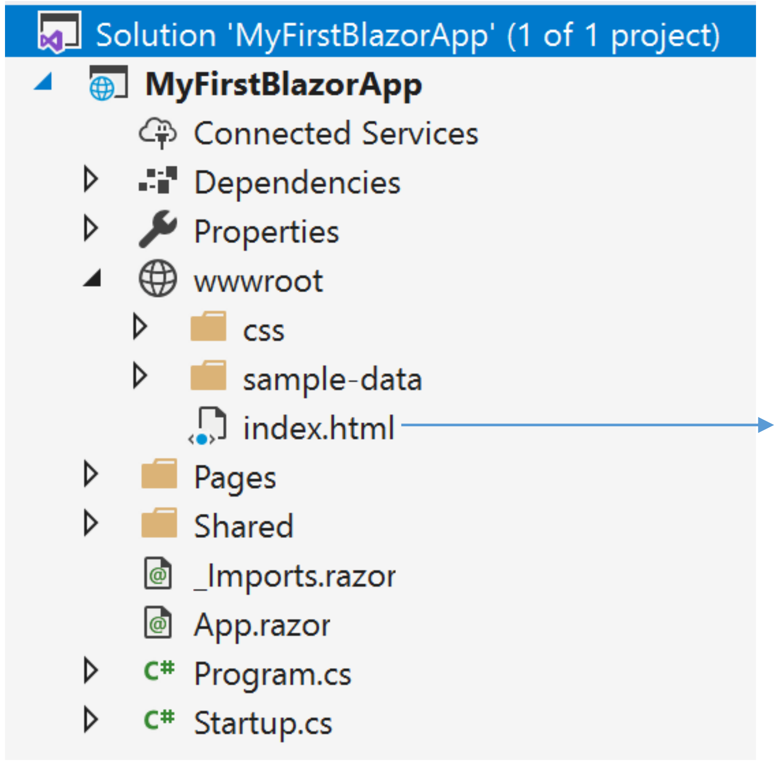
Module 2-1: Blazor

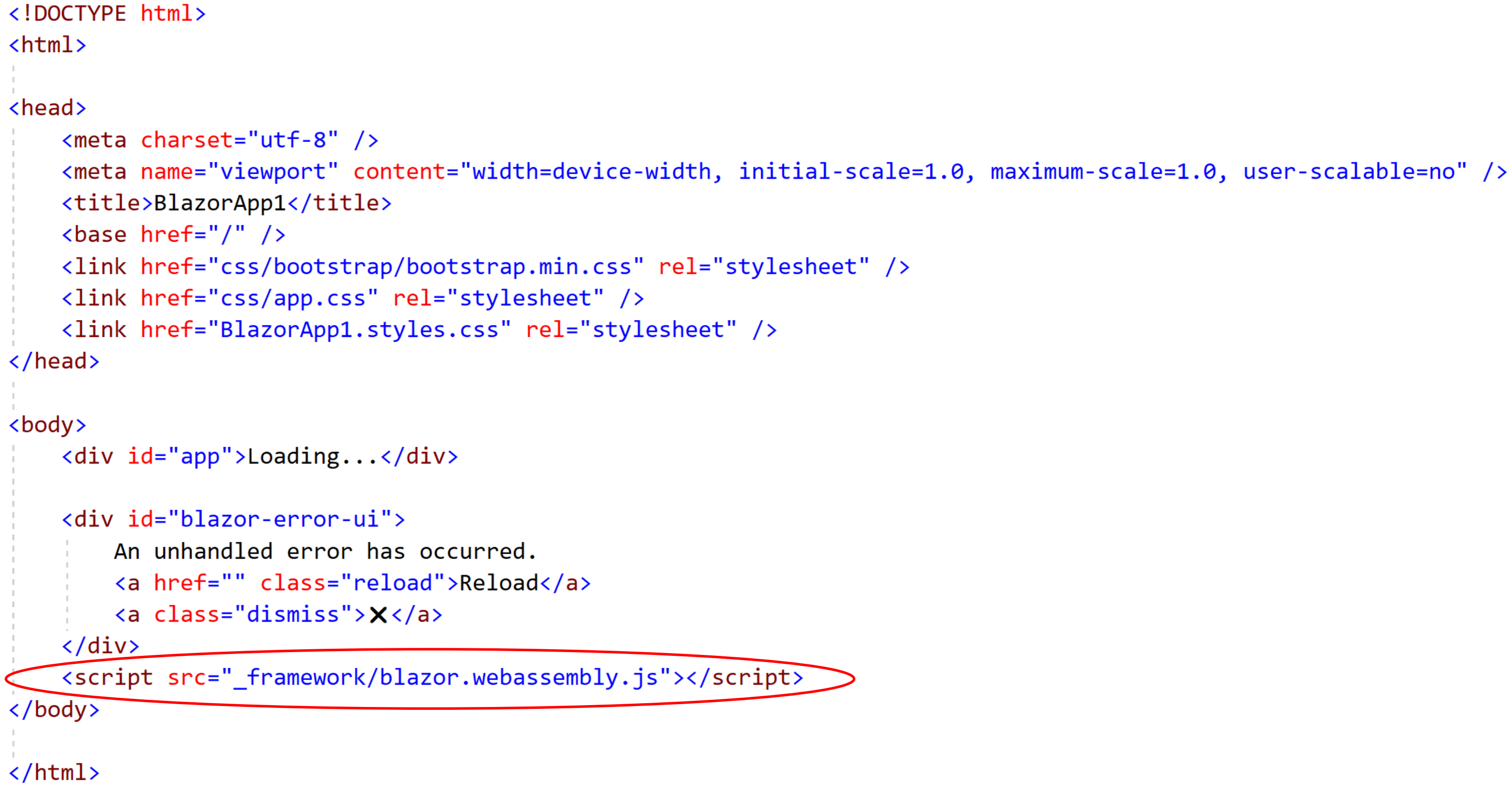
Section 2: Blazor

Lesson: Bootstrap Process

Blazor WebAssembly Bootstrap Process

- This script will install Blazor by downloading the compiled .net runtime (.wasm file) and your assemblies
  <script src="_framework/blazor.webassembly.js"></script>

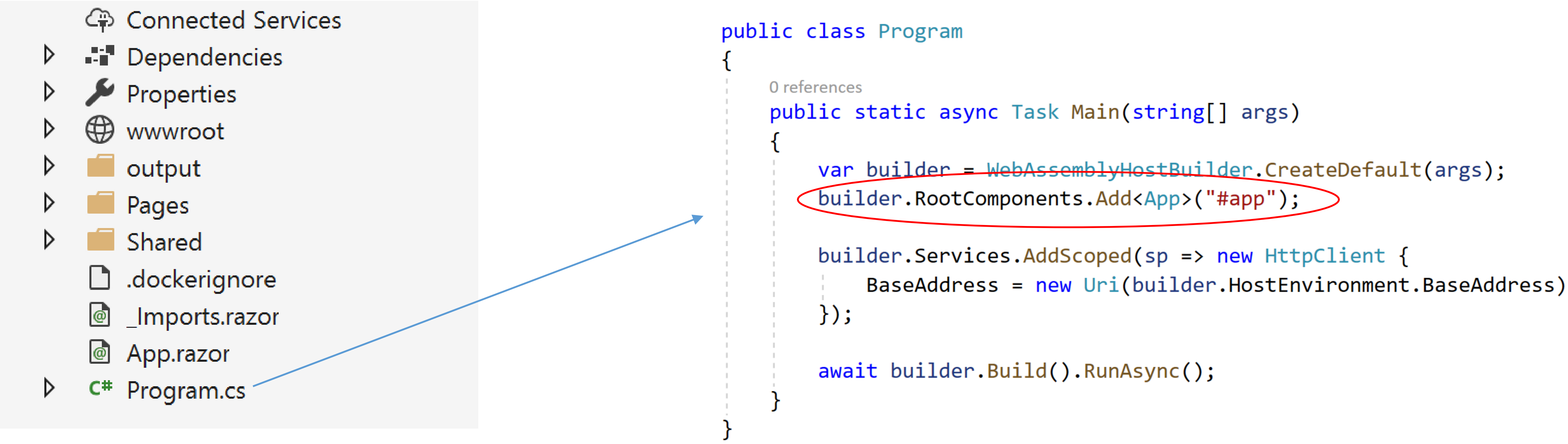




```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title>BlazorApp1</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
    <link href="BlazorApp1.styles.css" rel="stylesheet" />
</head>

<body>
    <div id="app">Loading...</div>

    <div id="blazor-error-ui">
        An unhandled error has occurred.
        <a href="" class="reload">Reload</a>
        <a class="dismiss">🗙</a>
    </div>
    <script src="_framework/blazor.webassembly.js"></script>
</body>

</html>
```

Blazor WebAssembly Bootstrap Process

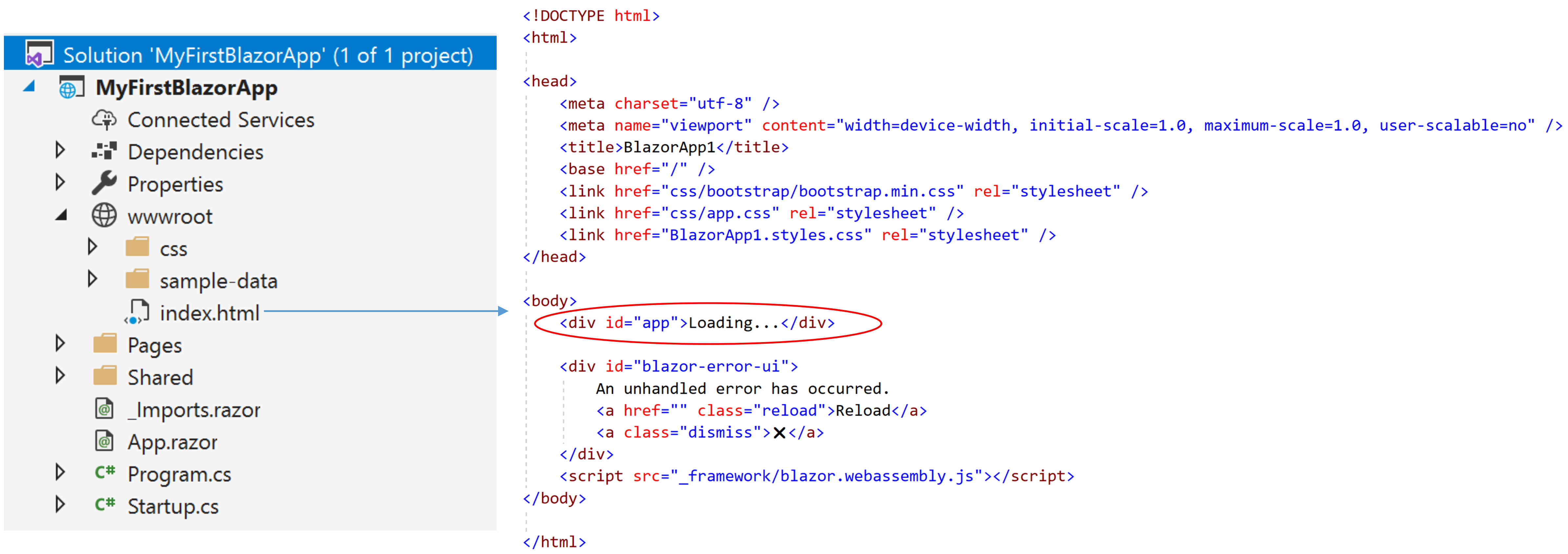
- The bootstrapping happens in the main method in the Program.cs file which loads the root component (App)
- It also associates App component with the div element with id #app inside index.html
- Blazor runtime places the component’s markup, which is normal HTML recognized by the browser, inside the aforementioned div element



```
public class Program
{
        0 references
        public static async Task Main(string[] args)
        {
            var builder = WebAssemblyHostBuilder.CreateDefault(args);
            builder.RootComponents.Add<App>("#app");

            builder.Services.AddScoped(sp => new HttpClient {
                BaseAddress = new Uri(builder.HostEnvironment.BaseAddress)
            });

            await builder.Build().RunAsync();
        }
}
```

Blazor WebAssembly Bootstrap Process

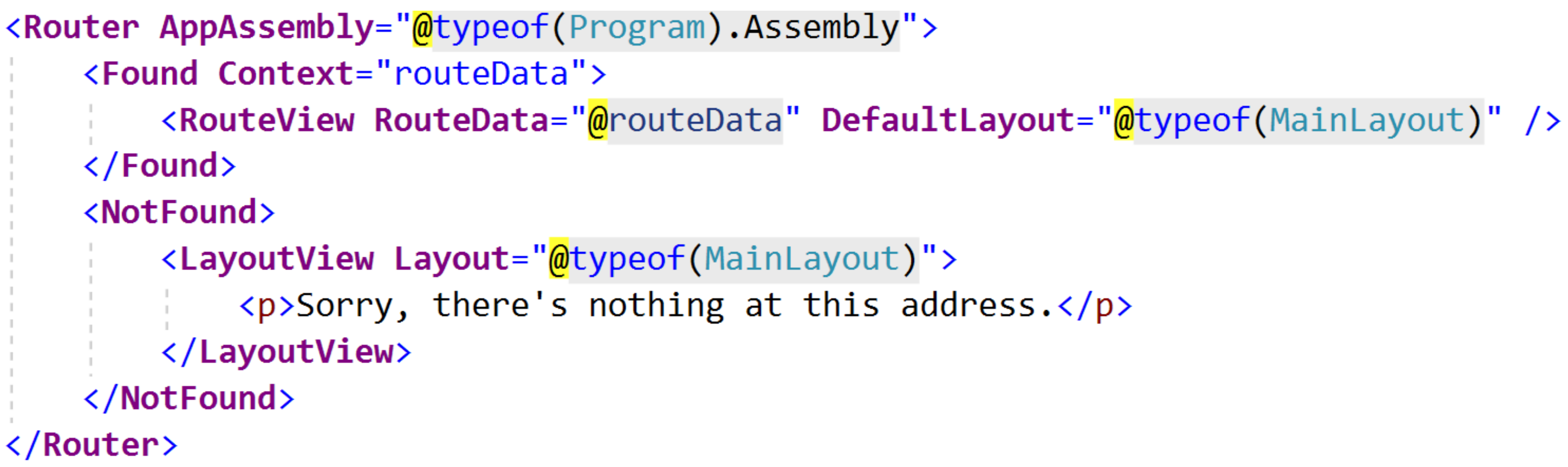
- This script will install Blazor by downloading the compiled .net runtime (.wasm file) and your assemblies
  <script src="_framework/blazor.webassembly.js"></script>



```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title>BlazorApp1</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
    <link href="BlazorApp1.styles.css" rel="stylesheet" />
</head>

<body>
    <div id="app">Loading...</div>

    <div id="blazor-error-ui">
        An unhandled error has occurred.
        <a href="" class="reload">Reload</a>
        <a class="dismiss">🗙</a>
    </div>
    <script src="_framework/blazor.webassembly.js"></script>
</body>

</html>
```

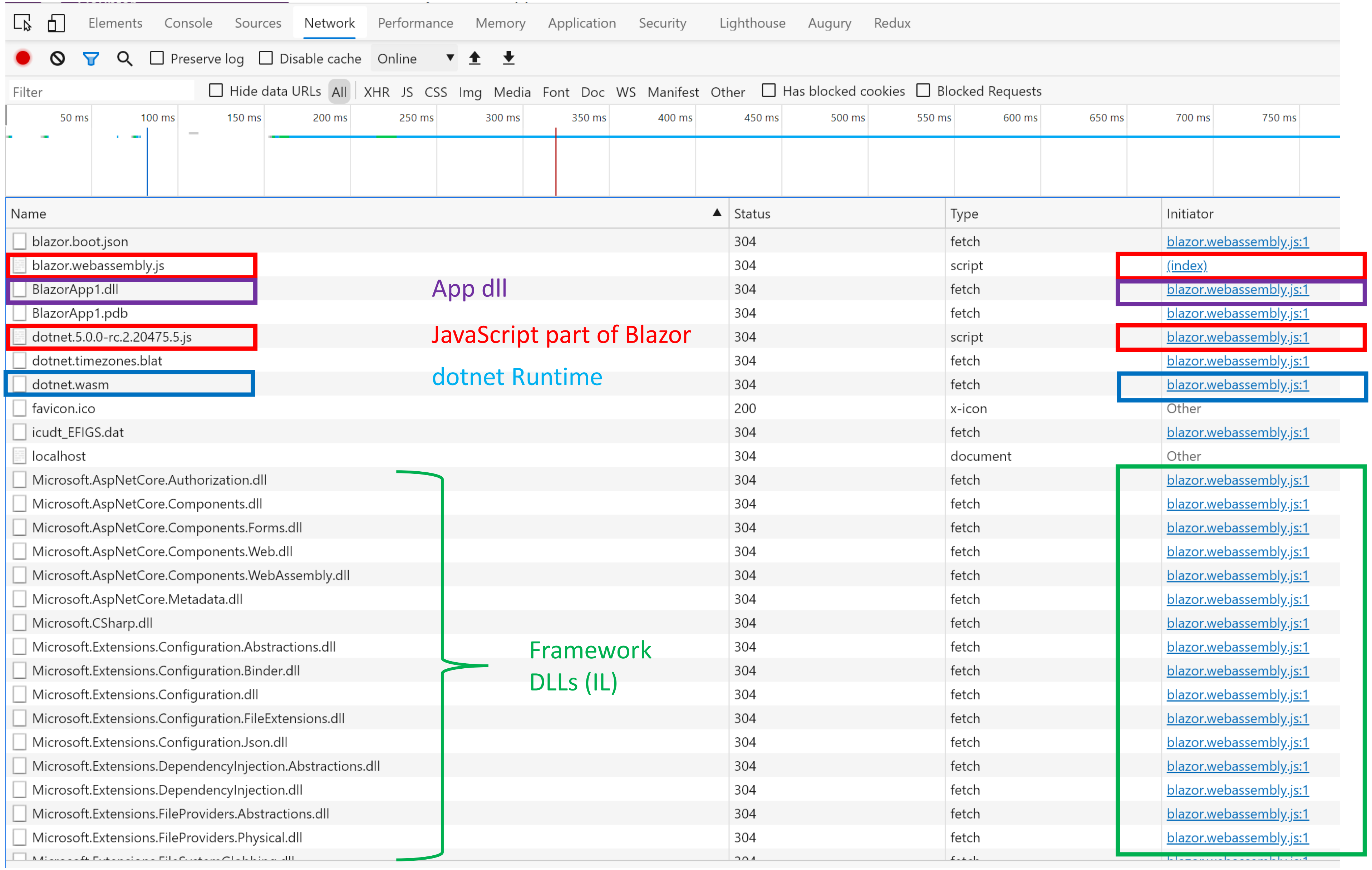
Blazor WebAssembly Bootstrap Process

- The app component is responsible for installing the router
- The router is responsible for loading a Blazor component depending on the URI in the browser
  - For example, if you browse to the / URI, the router will look for a component with a matching @page directive



```
<Router AppAssembly="@typeof(Program).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
    </Found>
    <NotFound>
        <LayoutView Layout="@typeof(MainLayout)">
            <p>Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

Blazor WebAssembly Bootstrap Process



Name	Status	Type	Initiator
blazor.boot.json	304	fetch	blazor.webassembly.js:1
blazor.webassembly.js	304	script	(index)
BlazorApp1.dll	304	fetch	blazor.webassembly.js:1
BlazorApp1.pdb	304	fetch	blazor.webassembly.js:1
dotnet.5.0.0-rc.2.20475.5.js	304	script	blazor.webassembly.js:1
dotnet.timezones.blat	304	fetch	blazor.webassembly.js:1
dotnet.wasm	304	fetch	blazor.webassembly.js:1
favicon.ico	200	x-icon	Other
icudt_EFIGS.dat	304	fetch	blazor.webassembly.js:1
localhost	304	document	Other
Microsoft.AspNetCore.Authorization.dll	304	fetch	blazor.webassembly.js:1
Microsoft.AspNetCore.Components.dll	304	fetch	blazor.webassembly.js:1
Microsoft.AspNetCore.Components.Forms.dll	304	fetch	blazor.webassembly.js:1
Microsoft.AspNetCore.Components.Web.dll	304	fetch	blazor.webassembly.js:1
Microsoft.AspNetCore.Components.WebAssembly.dll	304	fetch	blazor.webassembly.js:1
Microsoft.AspNetCore.Metadata.dll	304	fetch	blazor.webassembly.js:1
Microsoft.CSharp.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.Configuration.Abstractions.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.Configuration.Binder.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.Configuration.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.Configuration.FileExtensions.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.Configuration.Json.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.DependencyInjection.Abstractions.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.DependencyInjection.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.FileProviders.Abstractions.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.FileProviders.Physical.dll	304	fetch	blazor.webassembly.js:1
Microsoft.Extensions.FileSystemGlobbing.dll	304	fetch	blazor.webassembly.js:1

App dll

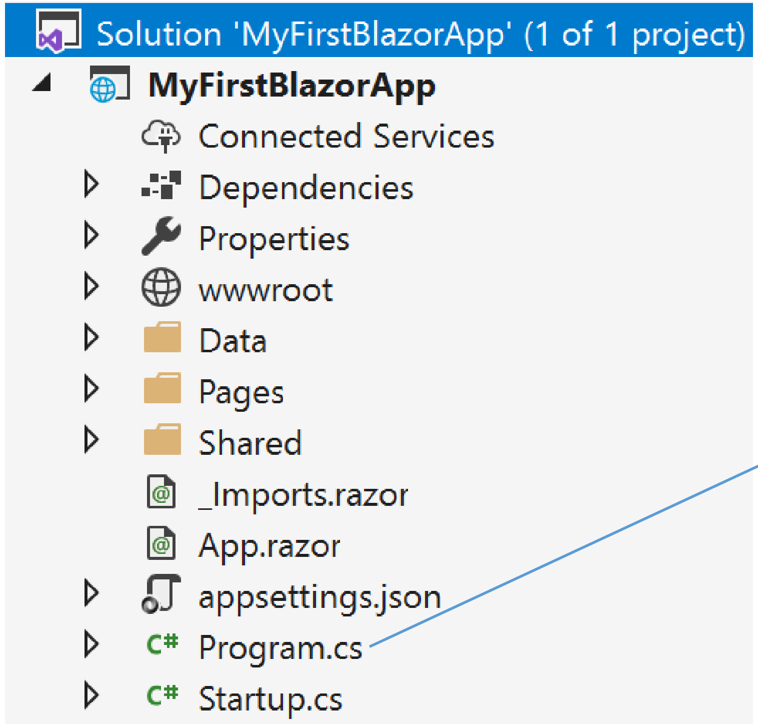
JavaScript part of Blazor

dotnet Runtime

Framework DLLs (IL)

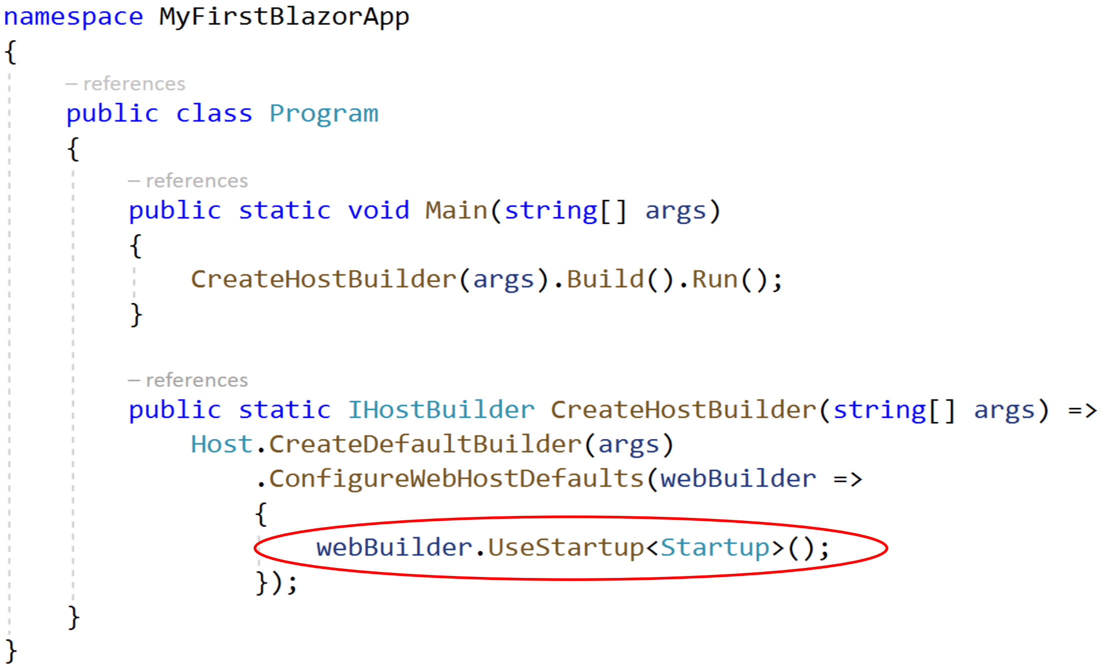
Server Side Blazor Bootstrap Process

- With the Blazor Server Side hosting model, the app is executed on the server from within an ASP.NET Core app



Solution 'MyFirstBlazorApp' (1 of 1 project)

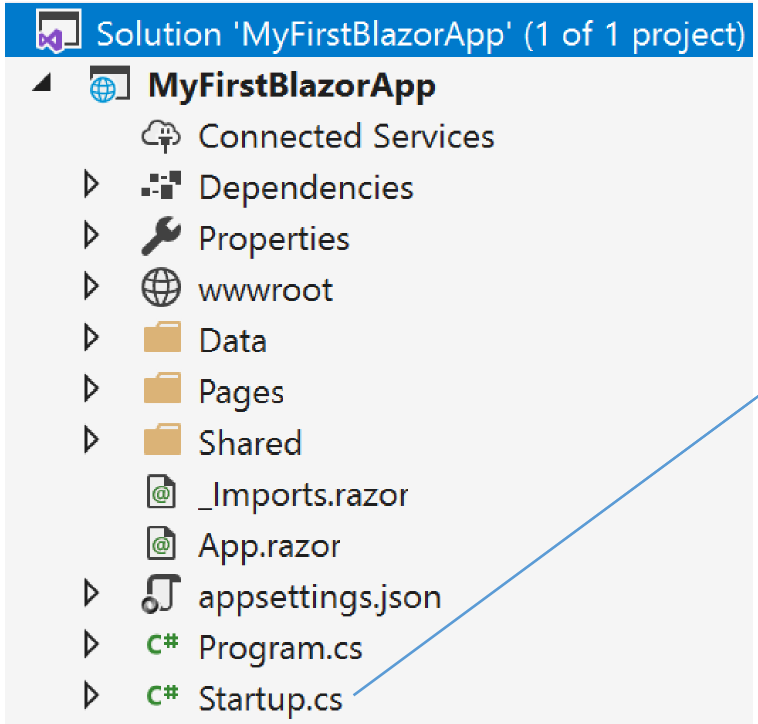
- **MyFirstBlazorApp**
  - Connected Services
  - Dependencies
  - Properties
  - wwwroot
  - Data
  - Pages
  - Shared
  - _Imports.razor
  - App.razor
  - appsettings.json
  - Program.cs
  - Startup.cs

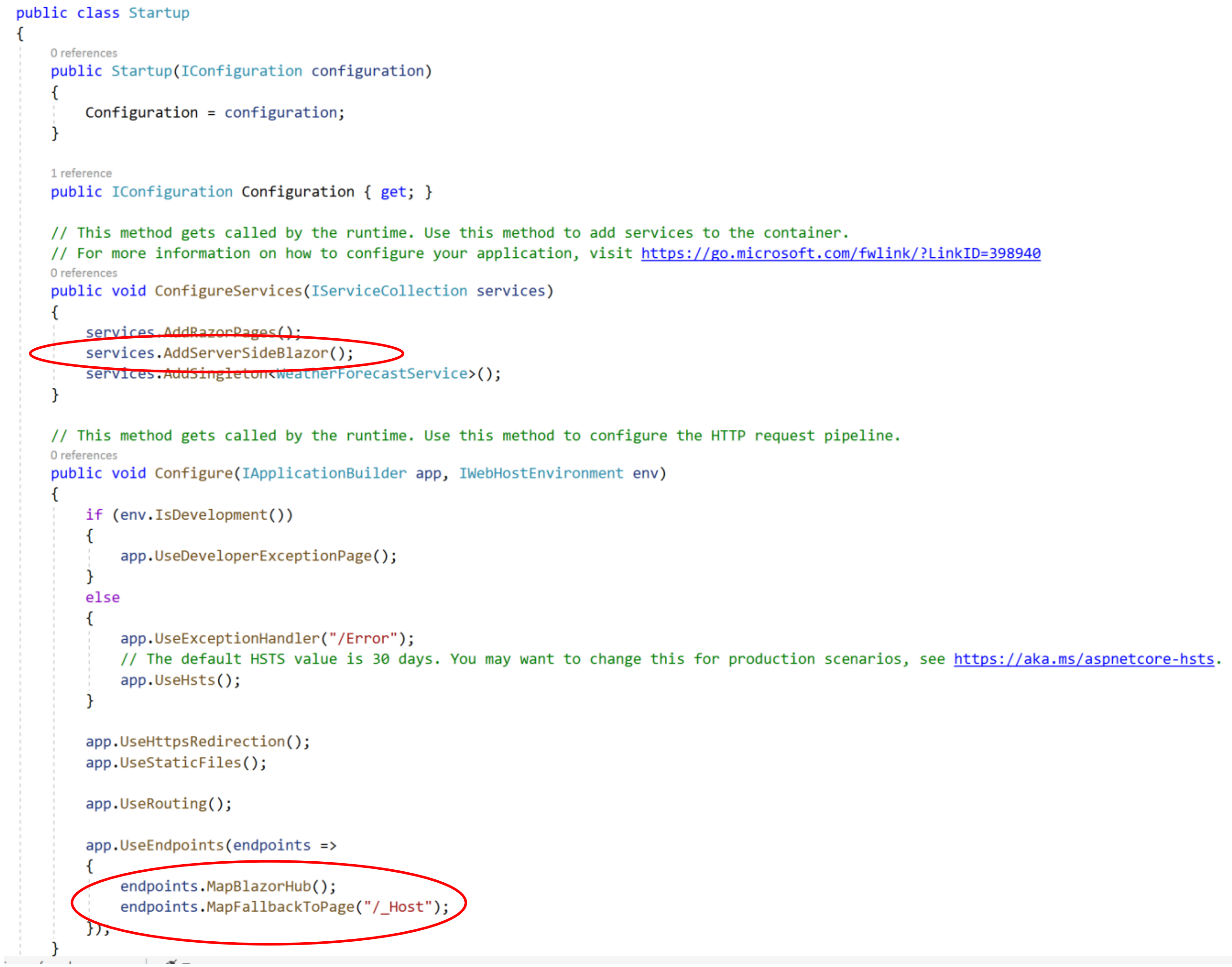


```
namespace MyFirstBlazorApp
{
    – references
    public class Program
    {
        – references
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        – references
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}
```

Server Side Blazor Bootstrap Process

- The ASP.NET Core app references the app's Startup class to add:
    - Server-side services
    - The app to the request handling pipeline

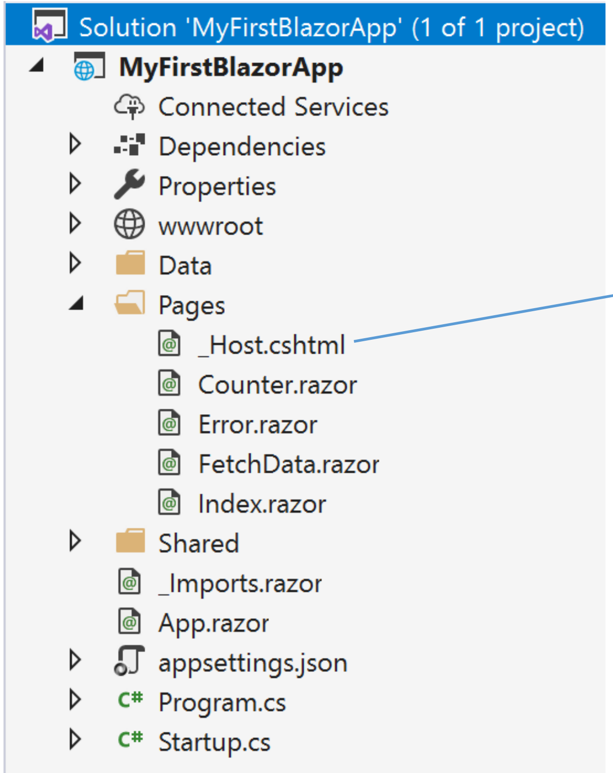




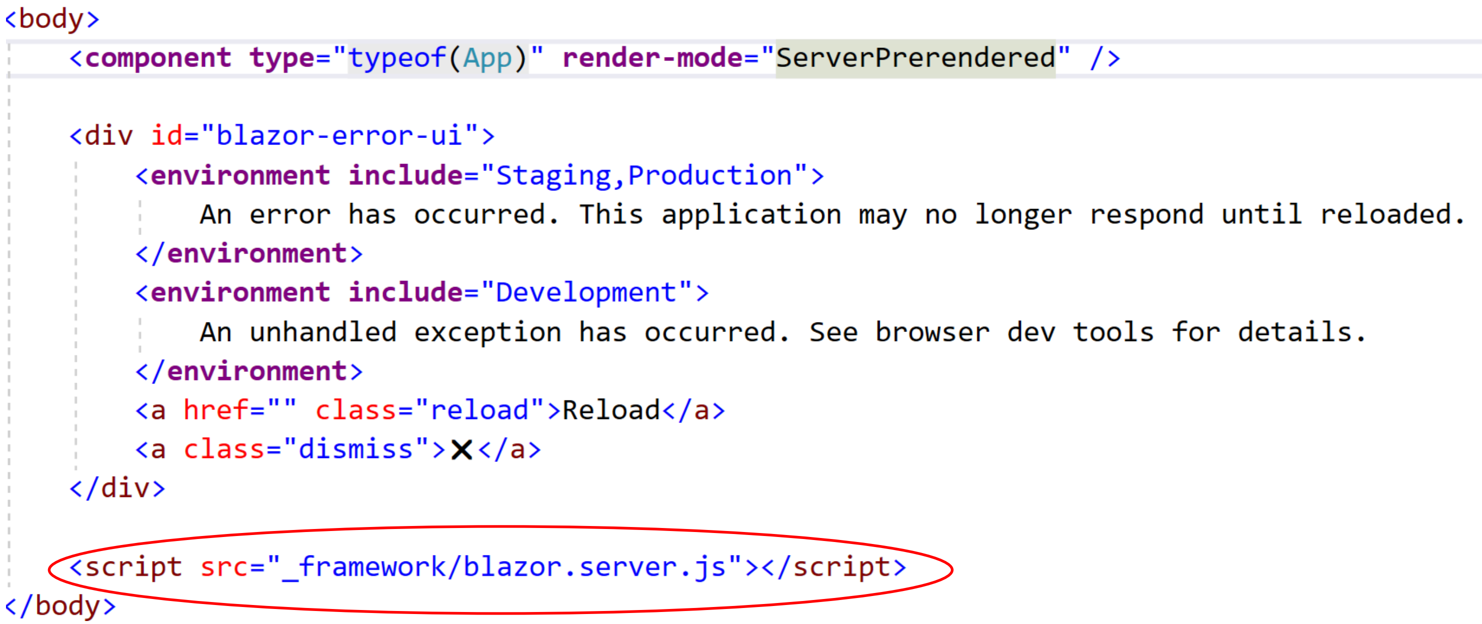
```
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddRazorPages();
        services.AddServerSideBlazor();
        services.AddSingleton<WeatherForecastService>();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    0 references
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Error");
            // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
            app.UseHsts();
        }

        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapBlazorHub();
            endpoints.MapFallbackToPage("/_Host");
        });
    }
```

Server Side Blazor Bootstrap Process

- The **blazor.server.js script** establishes the client connection
    - It's the app's responsibility to **persist and restore app state** as required (for example, in the event of a lost network connection)
    - Its **served from** an embedded resource in the **ASP.NET Core shared framework**

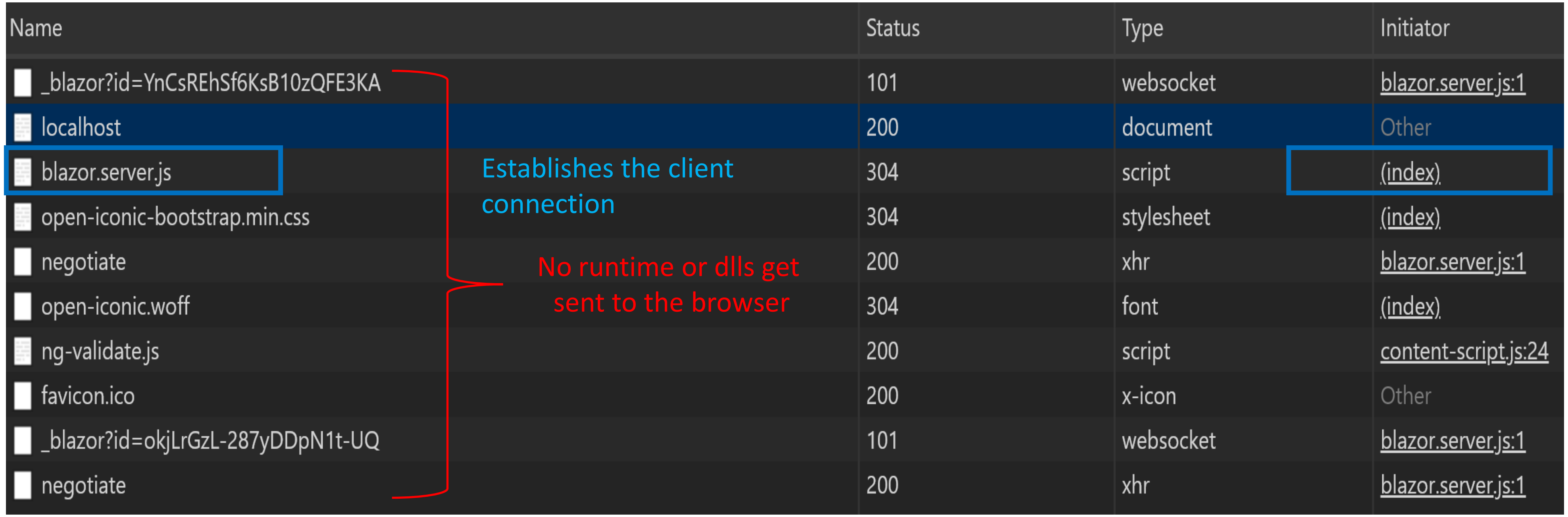


```
Solution 'MyFirstBlazorApp' (1 of 1 project)
◢ MyFirstBlazorApp
      Connected Services
  ▷   Dependencies
  ▷   Properties
  ▷   wwwroot
  ▷   Data
  ◢   Pages
          _Host.cshtml
          Counter.razor
          Error.razor
          FetchData.razor
          Index.razor
  ▷   Shared
      _Imports.razor
      App.razor
  ▷   appsettings.json
  ▷   Program.cs
  ▷   Startup.cs
```



```
<body>
    <component type="typeof(App)" render-mode="ServerPrerendered" />

    <div id="blazor-error-ui">
        <environment include="Staging,Production">
            An error has occurred. This application may no longer respond until reloaded.
        </environment>
        <environment include="Development">
            An unhandled exception has occurred. See browser dev tools for details.
        </environment>
        <a href="" class="reload">Reload</a>
        <a class="dismiss">✕</a>
    </div>

    <script src="_framework/blazor.server.js"></script>
</body>
```

Server-Side Blazor Bootstrap Process



Name	Status	Type	Initiator
_blazor?id=YnCsREhSf6KsB10zQFE3KA	101	websocket	blazor.server.js:1
localhost	200	document	Other
blazor.server.js	304	script	(index)
open-iconic-bootstrap.min.css	304	stylesheet	(index)
negotiate	200	xhr	blazor.server.js:1
open-iconic.woff	304	font	(index)
ng-validate.js	200	script	content-script.js:24
favicon.ico	200	x-icon	Other
_blazor?id=okjLrGzL-287yDDpN1t-UQ	101	websocket	blazor.server.js:1
negotiate	200	xhr	blazor.server.js:1

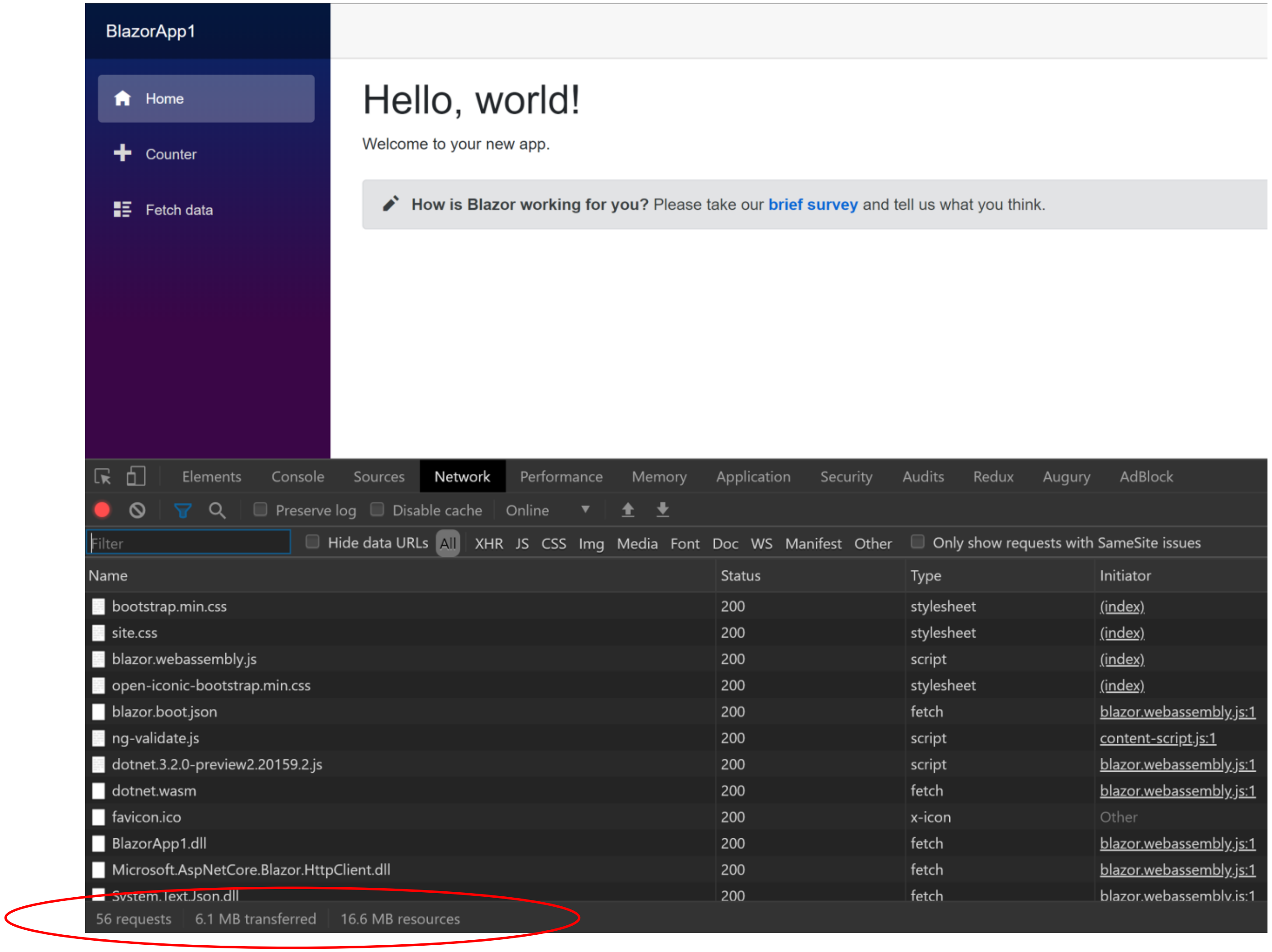
Establishes the client connection

No runtime or dlls get sent to the browser

Demo: Blazor WebAssembly Bootstrap Process

Blazor WebAssembly Framework Caching

- Initial Blazor WebAssembly app size is large as it has to download the application and framework dlls to the browser



Hello, world!

Welcome to your new app.

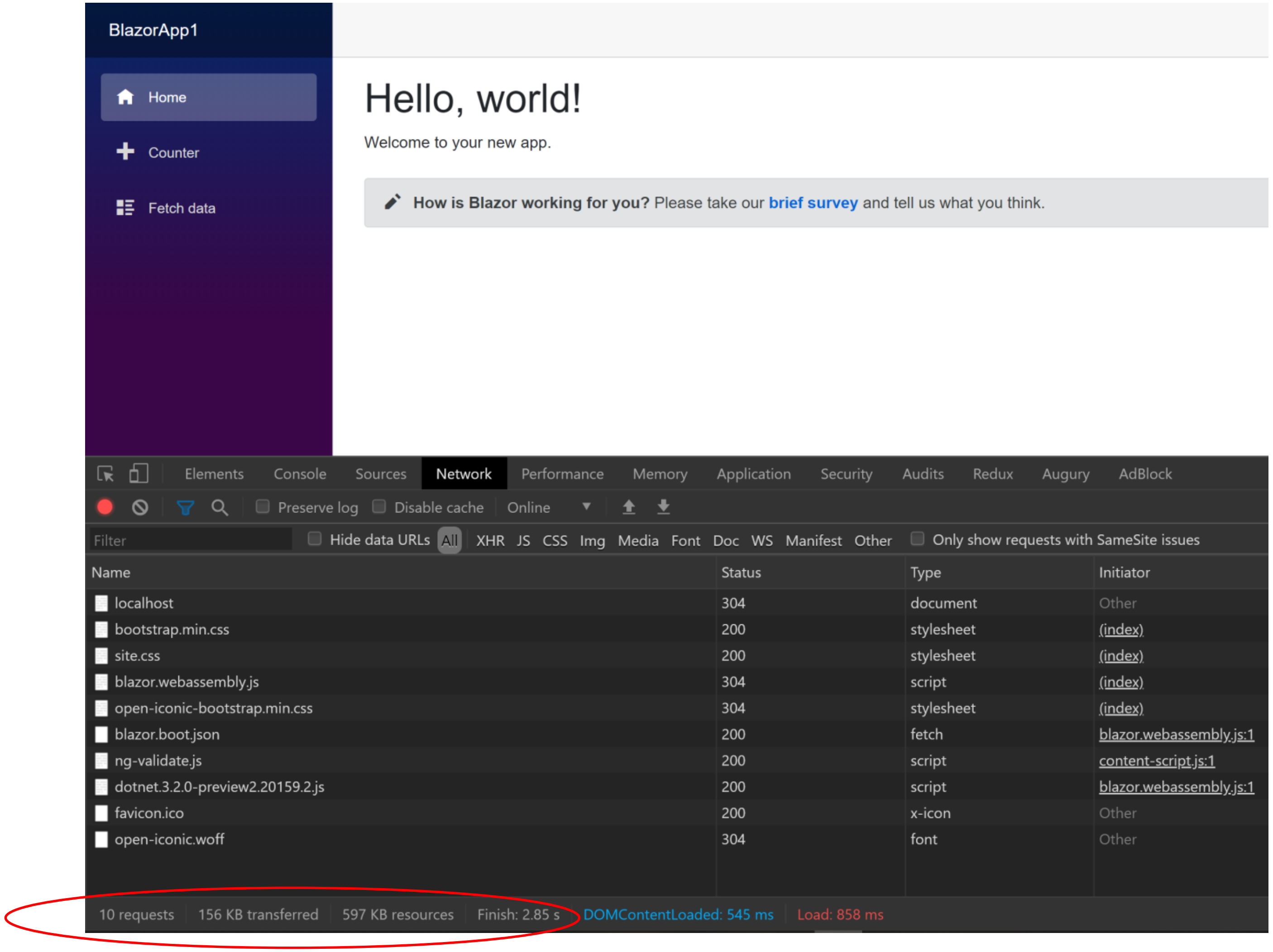
How is Blazor working for you? Please take our **brief survey** and tell us what you think.

Name	Status	Type	Initiator
bootstrap.min.css	200	stylesheet	(index)
site.css	200	stylesheet	(index)
blazor.webassembly.js	200	script	(index)
open-iconic-bootstrap.min.css	200	stylesheet	(index)
blazor.boot.json	200	fetch	blazor.webassembly.js:1
ng-validate.js	200	script	content-script.js:1
dotnet.3.2.0-preview2.20159.2.js	200	script	blazor.webassembly.js:1
dotnet.wasm	200	fetch	blazor.webassembly.js:1
favicon.ico	200	x-icon	Other
BlazorApp1.dll	200	fetch	blazor.webassembly.js:1
Microsoft.AspNetCore.Blazor.HttpClient.dll	200	fetch	blazor.webassembly.js:1
System.Text.Json.dll	200	fetch	blazor.webassembly.js:1

56 requests | 6.1 MB transferred | 16.6 MB resources

Blazor WebAssembly Framework Caching

- If you look at the network trace of what's being downloaded for a Blazor WebAssembly app after it's initially loaded you will notice that the size is extremely smaller



Hello, world!

Welcome to your new app.

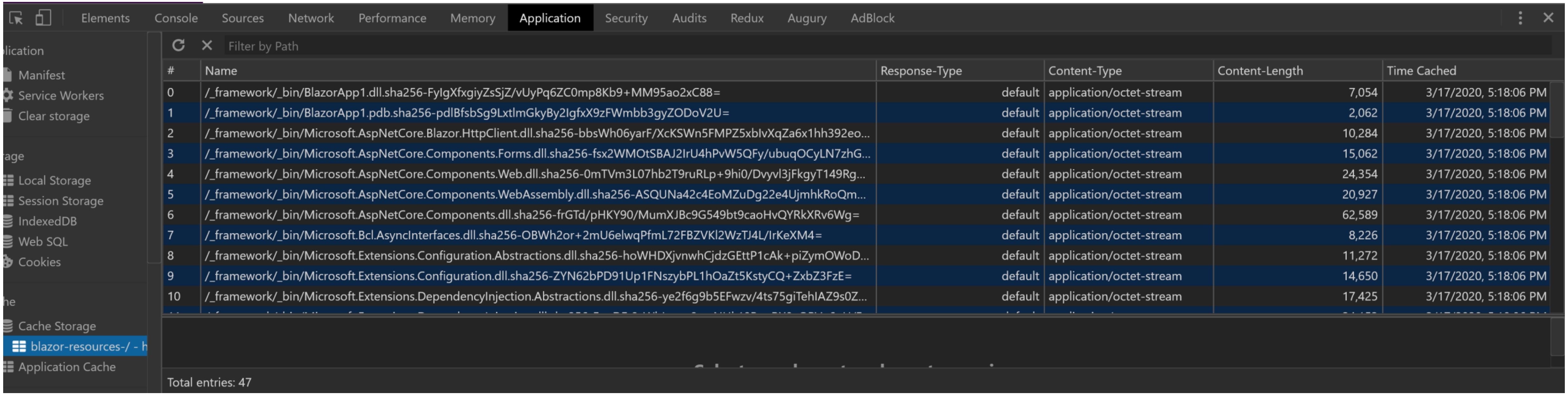
How is Blazor working for you? Please take our **brief survey** and tell us what you think.

Name	Status	Type	Initiator
localhost	304	document	Other
bootstrap.min.css	200	stylesheet	(index)
site.css	200	stylesheet	(index)
blazor.webassembly.js	304	script	(index)
open-iconic-bootstrap.min.css	304	stylesheet	(index)
blazor.boot.json	200	fetch	blazor.webassembly.js:1
ng-validate.js	200	script	content-script.js:1
dotnet.3.2.0-preview2.20159.2.js	200	script	blazor.webassembly.js:1
favicon.ico	200	x-icon	Other
open-iconic.woff	304	font	Other

10 requests | 156 KB transferred | 597 KB resources | Finish: 2.85 s | DOMContentLoaded: 545 ms | Load: 858 ms

Blazor WebAssembly Framework Caching

- When a Blazor WebAssembly app is initially loaded, the runtime and framework files are now stored in the browser cache storage
- When the app loads, it first uses the contents of the *blazor.boot.json* to check if it already has all of the runtime and framework files it needs in the cache. If it does, then no additional network requests are necessary

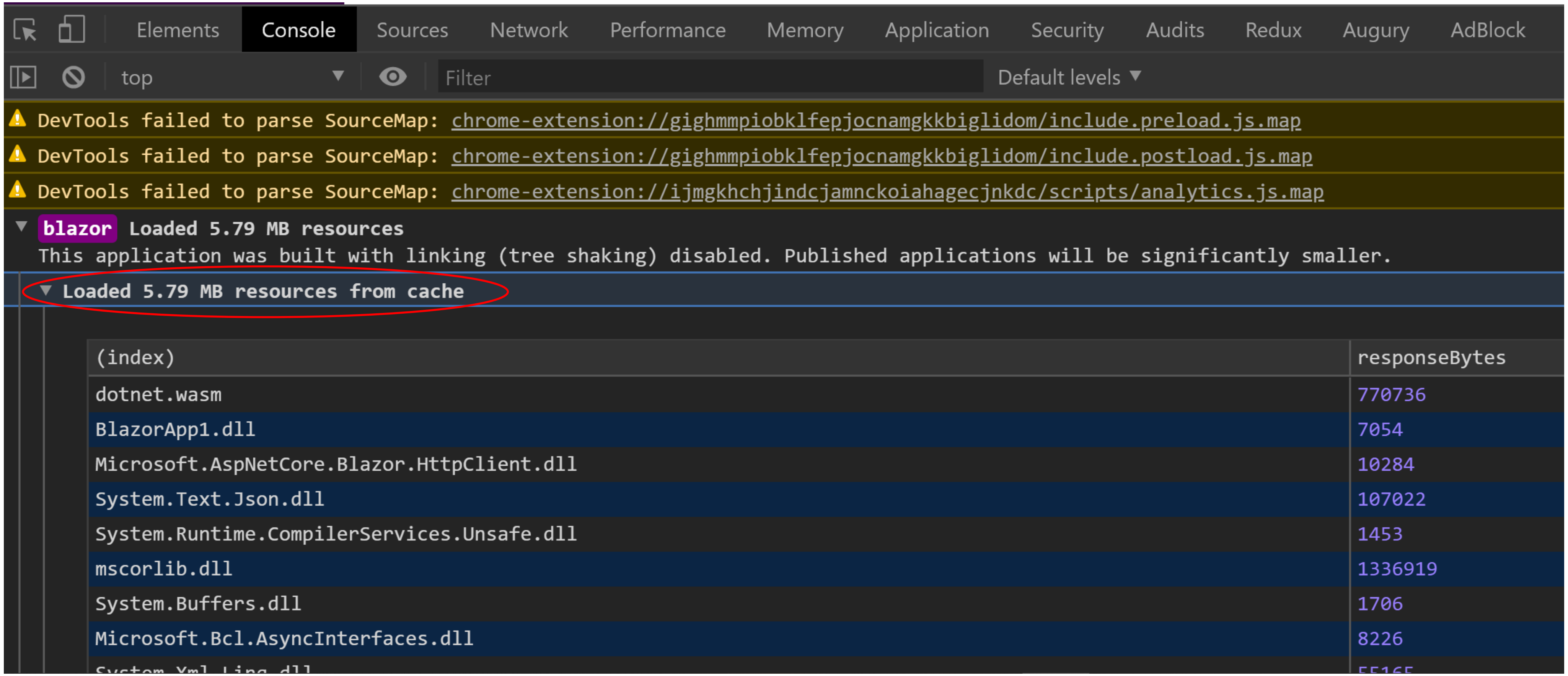


#	Name	Response-Type	Content-Type	Content-Length	Time Cached
0	/_framework/_bin/BlazorApp1.dll.sha256-FyIgXfxgiyZsSjZ/vUyPq6ZC0mp8Kb9+MM95ao2xC88=	default	application/octet-stream	7,054	3/17/2020, 5:18:06 PM
1	/_framework/_bin/BlazorApp1.pdb.sha256-pdlBfsbSg9LxtlmGkyBy2IgfxX9zFWmbb3gyZODoV2U=	default	application/octet-stream	2,062	3/17/2020, 5:18:06 PM
2	/_framework/_bin/Microsoft.AspNetCore.Blazor.HttpClient.dll.sha256-bbsWh06yarF/XcKSWn5FMPZ5xbIvXqZa6x1hh392eo...	default	application/octet-stream	10,284	3/17/2020, 5:18:06 PM
3	/_framework/_bin/Microsoft.AspNetCore.Components.Forms.dll.sha256-fsx2WMOtSBAJ2IrU4hPvW5QFy/ubuqOCyLN7zhG...	default	application/octet-stream	15,062	3/17/2020, 5:18:06 PM
4	/_framework/_bin/Microsoft.AspNetCore.Components.Web.dll.sha256-0mTVm3L07hb2T9ruRLp+9hi0/Dvyvl3jFkgyT149Rg...	default	application/octet-stream	24,354	3/17/2020, 5:18:06 PM
5	/_framework/_bin/Microsoft.AspNetCore.Components.WebAssembly.dll.sha256-ASQUNa42c4EoMZuDg22e4UjmhkRoQm...	default	application/octet-stream	20,927	3/17/2020, 5:18:06 PM
6	/_framework/_bin/Microsoft.AspNetCore.Components.dll.sha256-frGTd/pHKY90/MumXJBc9G549bt9caoHvQYRkXRv6Wg=	default	application/octet-stream	62,589	3/17/2020, 5:18:06 PM
7	/_framework/_bin/Microsoft.Bcl.AsyncInterfaces.dll.sha256-OBWh2or+2mU6elwqPfmL72FBZVKl2WzTJ4L/IrKeXM4=	default	application/octet-stream	8,226	3/17/2020, 5:18:06 PM
8	/_framework/_bin/Microsoft.Extensions.Configuration.Abstractions.dll.sha256-hoWHDXjvnwhCjdzGEttP1cAk+piZymOWoD...	default	application/octet-stream	11,272	3/17/2020, 5:18:06 PM
9	/_framework/_bin/Microsoft.Extensions.Configuration.dll.sha256-ZYN62bPD91Up1FNszybPL1hOaZt5KstyCQ+ZxbZ3FzE=	default	application/octet-stream	14,650	3/17/2020, 5:18:06 PM
10	/_framework/_bin/Microsoft.Extensions.DependencyInjection.Abstractions.dll.sha256-ye2f6g9b5EFwzv/4ts75giTehIAZ9s0Z...	default	application/octet-stream	17,425	3/17/2020, 5:18:06 PM

Total entries: 47

Blazor WebAssembly Framework Caching

- You can still see what the true size of the app is during development by checking the browser console



```
⚠ DevTools failed to parse SourceMap: chrome-extension://gighmmpiobklfepjocnamgkkbiglidom/include.preload.js.map
⚠ DevTools failed to parse SourceMap: chrome-extension://gighmmpiobklfepjocnamgkkbiglidom/include.postload.js.map
⚠ DevTools failed to parse SourceMap: chrome-extension://ijmgkhchjindcjamnckoiahagecjnkdc/scripts/analytics.js.map
▼ blazor Loaded 5.79 MB resources
  This application was built with linking (tree shaking) disabled. Published applications will be significantly smaller.
  ▼ Loaded 5.79 MB resources from cache
```

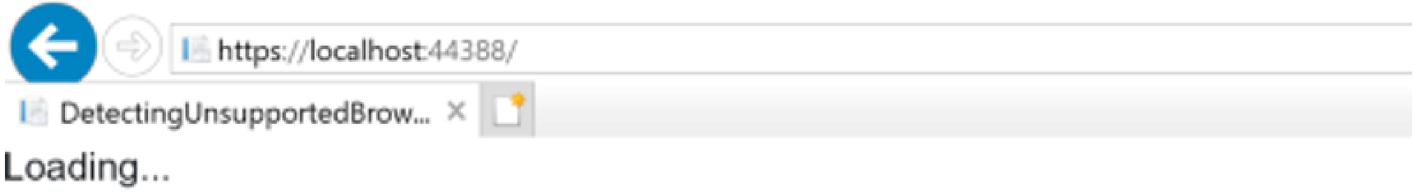
(index)	responseBytes
dotnet.wasm	770736
BlazorApp1.dll	7054
Microsoft.AspNetCore.Blazor.HttpClient.dll	10284
System.Text.Json.dll	107022
System.Runtime.CompilerServices.Unsafe.dll	1453
mscorlib.dll	1336919
System.Buffers.dll	1706
Microsoft.Bcl.AsyncInterfaces.dll	8226
System.Xml.Linq.dll	55165

Blazor WebAssembly Framework Caching

- What about invalidating the cache?
  - The cache is based on hashes of all of the files, which are then recorded in the blazor.boot.json file when the app is built. If any of the files change, the hashes should change and the cache will get invalidated

Detecting Unsupported Browsers

- What happens when you load a Blazor WebAssembly application Under older browsers?



https://localhost:44388/

DetectingUnsupportedBrow...

Loading...

Demo: Detecting Unsupported Browsers

Module Summary

- In this module, you learned about:
  - WebAssembly
  - Blazor Hosting Models
  - Blazor Bootstrap Process
  - Blazor WebAssembly Framework Caching

References

- Microsoft Docs

Microsoft