Introduction to the Blazor Framework – Episode 1

DEBASIS SAHA

Solution Architect, MCT, MVP

DEBASIS SAHA

Solution Architect, MCT, MVP

Author, Speaker, Blogger

Currently I am working as a Solution Architect at WeFiveSoft Pvt Ltd, Pune. I am a Microsoft Certified Trainer & C# Corner MVP. Having 12+ years of experience in the IT Industry. I am always been a great fan of Microsoft Technologies and love working on them. I have expertise in Asp.Net, .NET Core, MVC, C#, Azure, Asp.Net Core, SQL Server, ReactJs, Angular, and NoSQL Databases like MongoDB, CosmosDB, etc. He loves to write articles about these technologies.

debasis.ds@outlook.com

https://www.linkedin.com/in/sahadebasis/

@debasiskolsaha

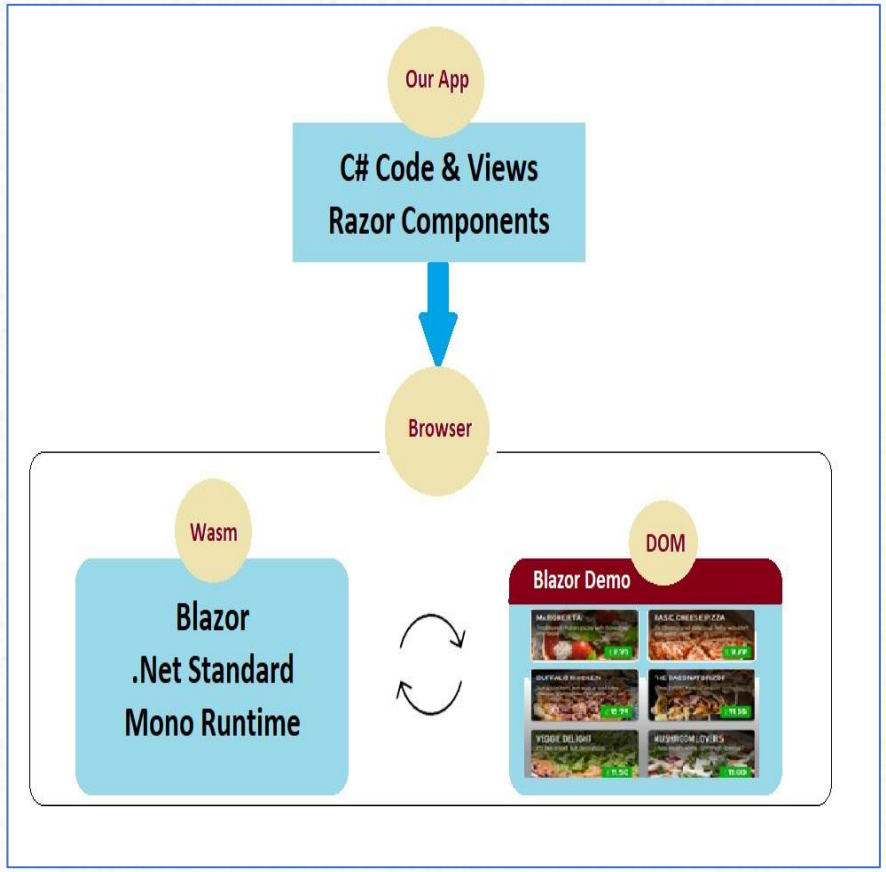
Agenda

- What is Blazor?
- What is Web Assembly?
- Overview of Different Blazor Hosting Models
- Blazor Web Assembly vs. Server Side
- Pros and Cons of Blazor Framework
- When can use the Blazor
- Blazor Projects structure in Visual Studio 2022

What is Blazor Framework?

What is Blazor?

- Microsoft introduced the **Blazor Framework** in the year of **2018**.
- Blazor is an open-source .Net web framework run on the Client-Side.
- Blazor combines two words – **Browser** and **Razor** (used as .Net HTML View generate engine).
- Can use **C#, HTML,** and **CSS** to create the **web-based UI** components
- Blazor executes the **Razor Views directly** on the **client's browser**.
- Can build **server-side** and **client-side** code with the help of the C#.
- Can use the **same class or code** as a **shared code or library**.



What is Web Assembly?

- WebAssembly is treated as a **low-level-based assembly**.
- Therefore, it is also known as a **web standard**.
- Can **execute** the web page's code at the **same speed**, just like **native machine code**.
- The **high-level language** used to develop the web application is **converted into WebAssembly** and then runs like a native code speed in the browser.
- We can now use the **conventional server-side languages** like C# and F# to develop any web application which can be run into the browser directly.

Web Assembly Browser Support

Browser	From Version
Chrome	57
Chrome for Android	74
Edge	16
Firefox	52
Firefox for Android	66
Opera	44
Opera for Mobile	46
Safari	11
iOS Safari	11
Samsung Internet	7.2
Android Browser	67

Blazor is Open-Source

- The source code is owned by **The .NET Foundation**, a non-profit organization. It is created for the purpose of supporting open-source projects based around the .NET framework.
- According to the .NET foundation, currently it is supported by
  - **3,700 companies**
  - **61,000 contributors**
  - **56 Active Projects**
  - **462 Resource**

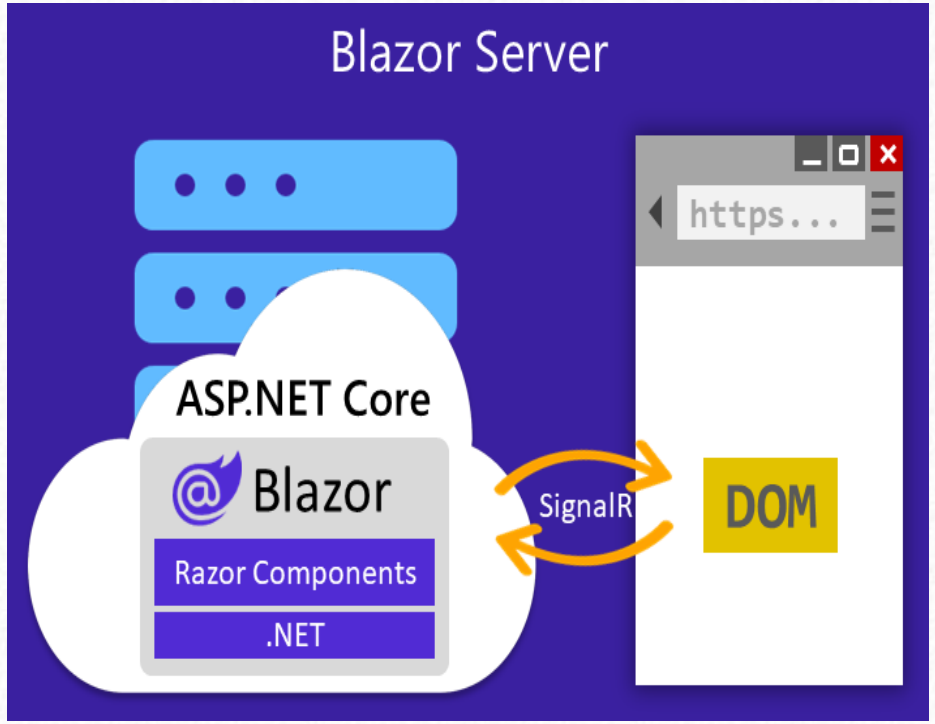
Blazor Hosting Model

Blazor Hosting Model

- Currently Blazor has **three** hosting models –
  - **Blazor Server-Side** – Released in September 2019.
  - **Blazor WebAssembly –** Release in May 2020.
  - **Blazor Hybrid** - Release with .Net 6.0 in November 2021.

Blazor Server Side

- Browser is considered as a **thin client** instead of **WebAssembly**.
- Code is developed and executed on the server side with the help of the **.Net Core Runtime**.
- Need to bootstrap the **SignalR library** to establish the connection on the client side.
- Can **send or receive** the **asynchronous events** between the client and server sides.
- The **application state** on the server side linked with each **thin client** is known as a **circuit**.
- **Circuits** are not tightly coupled with any specific network system.



Blazor Server Side

Pros

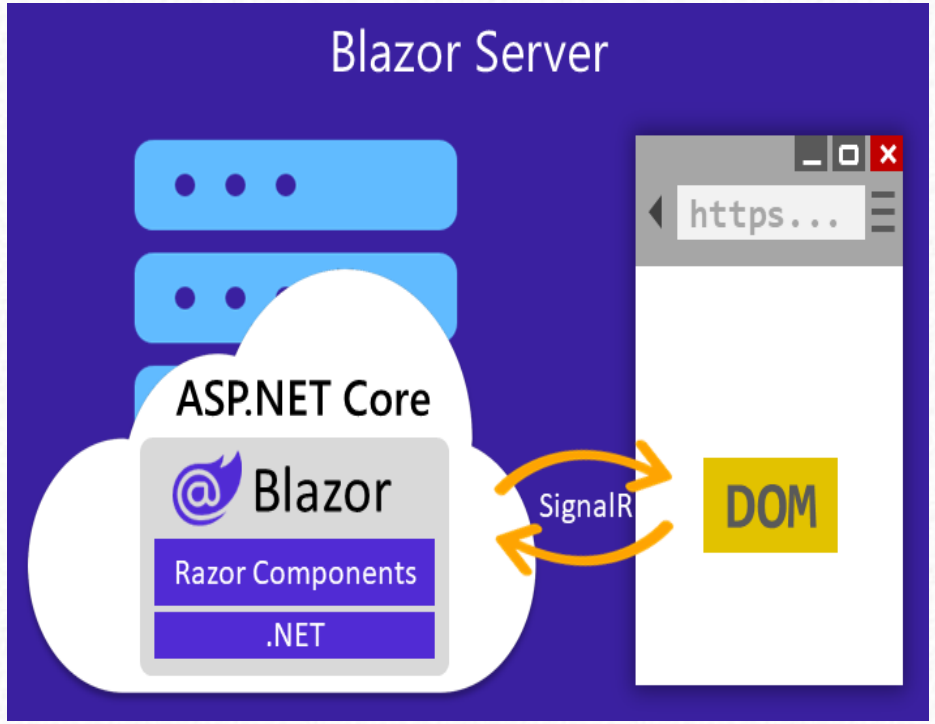
- The **download size** of the Blazor Server App is smaller compared to others
- We can use the server's full features of the **.Net Core APIs**.
- We can use **.Net Core** on the **server side** to execute the application. So we can easily **debug the application** whenever required.
- Blazor Server app is suitable if we are **trying to run** the application in such **browsers where WebAssembly is not supported.**

Cons

- Blazor Server app **does not provide any offline support**. So, if the client connection fails, the application will stop working.
- **Scaling** the application for **many users** always requires **multiple client states and client** connections.
- An **Asp.Net Core-based web server** is required to host the apps.
- **Serverless implementation**, like CDN, is impossible

Blazor Web Assembly

- **WASM-based application** mainly runs on the client-side browser with the help of **WebAssembly-based .NET runtime**.
- On First Time use, the **Blazor app**, its related **dependencies**, and **the .Net runtime** are **downloaded** in the clients' browser.
- The application is always **executed on the browser using the UI thread** and handles the **event-related** processes.
- CSS, JS, image files, etc., are considered **static files**.
- If we make the application without the **support of the Asp.Net Core back-end** app for its server-type files as **Blazor WebAssembly** app.
- Also can create the application **with the back-end** app options to **serve its server-side** files as **Hosted Blazor WebAssembly** app.



Blazor Web Assembly

Pros

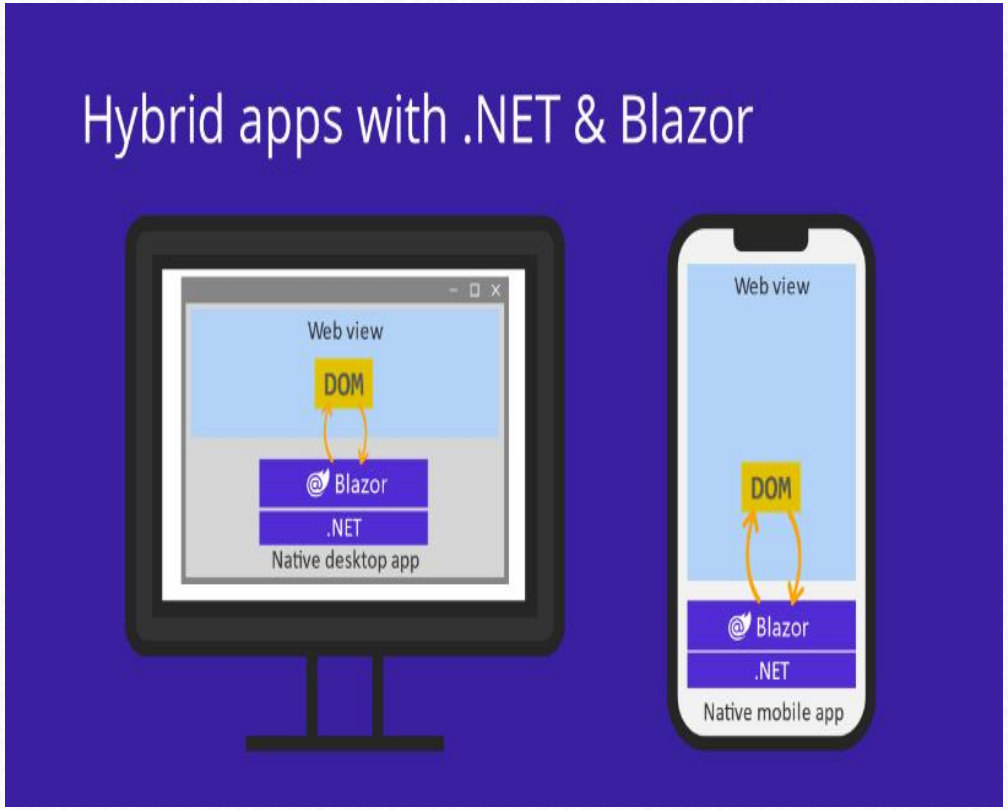
- There is **no dependency** once the app is **downloaded** into the **client machine** from the server. So, if the **server goes offline**, the app will continue working.
- **Asp.Net Core web server** is not required to host the application.
- Can use **serverless deployment** mechanism like **CDN** to host the application.
- We can use the **client's resources and capabilities** fully

Cons

- Application is **dependent** on the **capability of the browser**.
- When the **application is opened** for the **First Time**, it downloads the **entire app-related resources** along with runtime libraries and dependencies.
- As per the **application size**, it can take **longer to load the application** for the first time.
- As the application entirely depends on the **client's machine**, proper **software and hardware configuration** are required for the client's machine.

Blazor Hybrid

- Can develop the **native application** with the help of the Blazor Hybrid Hosting Model.
- Can use **Razor Components directly** within the native app with the help of .Net code,
- Can **render those components** as the **embedded web view control** using HTML and CSS.
- Can use different types of **.Net-based native application frameworks** like MAUI, Windows Form, and WPF.
- Can also use the **BlazorWebView controls** for implementing the Razor components within the app to build this framework.
- With the help of **Blazor along with .Net MAUI**, we can develop any type of cross-platform Blazor Hybrid application for both mobile and desktop.



Blazor Hybrid

Pros

- We can **reuse the existing components**, which can be shared across the **web, mobile, and desktop** environments.
- Can use the **existing development skills, experience, and resources** on Blazor Framework.
- Can use the **complete native functionality** of the device.

Cons

- Need to develop and maintain **separate native client apps** for **every target platform** in this hosting model.
- Native Client Apps require **extended time to fetch, download and install compared** to accessing any web app in the browser.

Blazor WebAssembly vs Blazor Server

Blazor Web Assembly	Blazor Server App
WebAssembly or wasm executes in the browser of the client.	Blazor Server app executes the C# code on the server side.
The client app uses separate .Net runtime in the browser. So we can use the different Web API servers for the communication or any other back-end framework for the back-end part.	It used 100% .net runtime on the server side. And it used a JS-based framework by the client to establish with the server
Blazor WASM is much more agnostic towards the server-side code.	It can use frameworks or protocols like web API, gPRC, signalR, etc., to communicate with the server.
For the first request on the client side, the wasm app downloads the related CLR, assemblies, and other files like JS, CSS, etc.	In the case of server apps, JavaScript hooks are used to access the DOM element in the browser.
Blazor WebAssembly is a front-end-based technology. So, when we compile it, we will get a similar web pack, just like a React Application.	Blazor Server app does not compile anything on the client side.
The app's download size is larger.	The app's download size is smaller.

Benefits of Blazor Framework

- Helps us to **develop single-page application**s just like React, Vue, and Angular.
- It is **fast, reusable, and open source**, providing the way for tremendous support from the community.
- **Server-side** rendering
- Provide **component-based architecture** for building reusable UI sections.
- Forms and validation
- JavaScript interop
- Dependency injection
- Routing
- Publishing and app size reducing
- Rich IntelliSense and tooling
- Live reloading in the browser during development
- Can run the application on older browsers also by using asm.js.

When we can use Blazor

- If the target users use the **browser that supports WebAssembly**, then we can use the **Blazor Framework**.
- We can also use **Blazor for Desktop or Mobile** except for the web. However, this functionality is still in preview. But we need to use either Electron or WebWindows in this scenario.
- We can use the **Blazor Server app** for any type of **intranet or internal** application.
- If the application **does not require many concurrent users**, then we can use the **Blazor Server App**.
- If the application requires a **large volume of concurrent users at a time**, then **Blazor WebAssembly** is the best option.
- Also, if we want to implement the **offline or PWA-based** functionality, **Blazor WebAssembly** needs to be selected.

Demo

Create Sample Project on Blazor using Visual Studio 2022

You can connect with me through –


debasis.ds@outlook.com


https://www.linkedin.com/in/sahadebasis


https://github.com/debasis-saha


@debasiskolsaha

Thank You

Questions