Blazor Hosting Models

---

1. Hosting Models in Blazor – Big Picture

Blazor is a **Single Page Application (SPA) framework** built on .NET that allows you to write client-side UI using **C# and Razor instead of JavaScript**.

Blazor supports **following hosting models**:

1. **Blazor Server**
2. **Blazor WebAssembly (WASM)**
3. **Blazor Hybrid (Blazor + .NET MAUI)**

Each model differs mainly in:

- **Where the code executes**
- **How UI events are processed**
- **How the browser communicates with .NET**

---

2. Introduction to Blazor Server

What is Blazor Server?

Blazor Server runs **all application code on the server**.
The browser only renders HTML and sends UI events (clicks, inputs) back to the server.

Key Concept:

UI logic executes on the server, not in the browser.

How It Works

1. User requests the application.
2. Server sends a minimal HTML page.
3. A **persistent SignalR connection** is established.
4. UI events are sent to the server.
5. Server processes events and sends **DOM diffs** back to the browser.

Characteristics

Aspect	Description
Execution	Server
Client requirement	Any modern browser
Download size	Very small
Initial load	Very fast
Internet dependency	Mandatory

---

3. Blazor Server Communication with SignalR

What is SignalR?

SignalR is a real-time communication framework in ASP.NET Core that enables bi-directional, persistent communication between a client (browser, mobile app) and a server.

Why SignalR?

Blazor Server uses **SignalR** to maintain a **real-time, bi-directional connection** between browser and server.

SignalR Responsibilities

- Tracks UI state per user
- Sends user events to server
- Sends UI updates (DOM diffs) to client

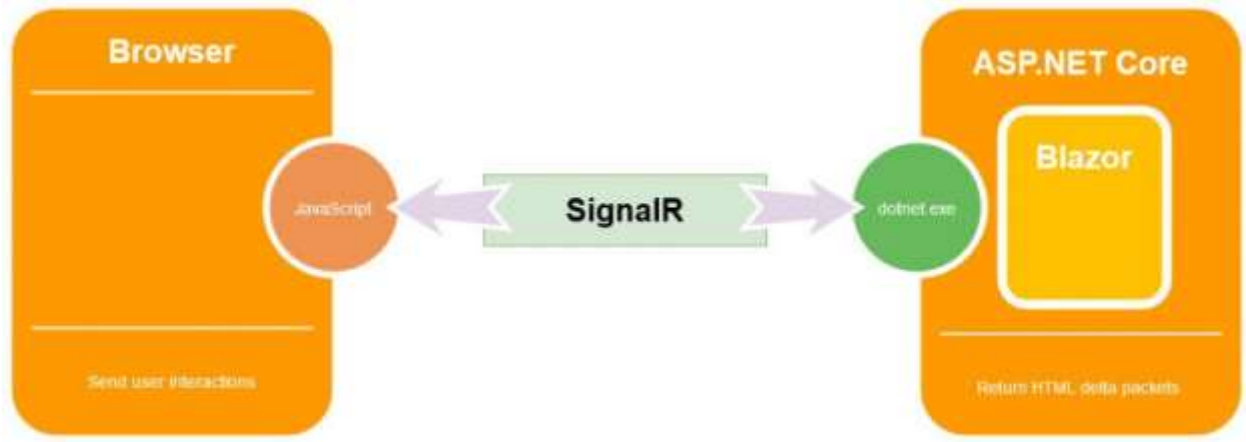
Advantages

- Centralized business logic
- Easy authentication & authorization
- Strong security (code never reaches client)

Challenges

- High concurrent users = high server memory
- Network latency affects UI responsiveness

- Not ideal for unreliable networks



---

4. Introduction to Blazor WebAssembly

What is Blazor WebAssembly?

Blazor WebAssembly runs **entirely in the browser** using **WebAssembly (WASM)**.

Key Concept:

Your .NET application is downloaded and executed inside the browser.

How It Works

1. Browser downloads:
    - .NET runtime (WASM)
    - Application DLLs
2. Application runs locally in browser sandbox
3. UI logic executes client-side
4. API calls are made using HTTP

Characteristics

Aspect	**Description**
Execution	Browser

Aspect	Description
Download size	Large
Initial load	Slower
Offline support	Possible
Server load	Low
Scalability	Excellent

---

5. Blazor WebAssembly Communication Using WebAssembly

Communication Model

Blazor WASM **does not use SignalR by default**.

Instead, it uses:

- HttpClient
- REST APIs
- gRPC (with limitations)
- JavaScript Interop

Execution Flow

1. User interacts with UI
2. C# code runs inside browser
3. API calls are sent to backend services
4. Responses update UI locally

Browser Sandbox Constraints

- No direct file system access
- No direct OS access
- Limited threading
- Depends on browser security model

---

6. Blazor WebAssembly vs Blazor Server

Comparison Table

Feature	**Blazor Server**	**Blazor WebAssembly**
Execution	Server	Browser
Initial Load	Very fast	Slower
Runtime Location	Server	Client
Internet	Required	Optional
Scalability	Moderate	High
Security	Strong	Code exposed
Offline	Not supported	Supported
Hosting Cost	Higher	Lower

When to Choose What?

Choose Blazor Server when:

- Enterprise internal apps
- Real-time dashboards
- Strong security requirements
- Fast development & low client constraints

Choose Blazor WebAssembly when:

- Public-facing applications
- Offline support required
- High scalability needed

- Client-heavy interactions

---

7. Blazor Hybrid / .NET MAUI

What is Blazor Hybrid?

Blazor Hybrid embeds Blazor UI inside **native applications** using **.NET MAUI**.

Supported platforms:

- Windows
- macOS
- Android
- iOS

How It Works

- Blazor UI runs inside a **WebView**
- .NET code runs **natively**, not in browser
- Full access to:

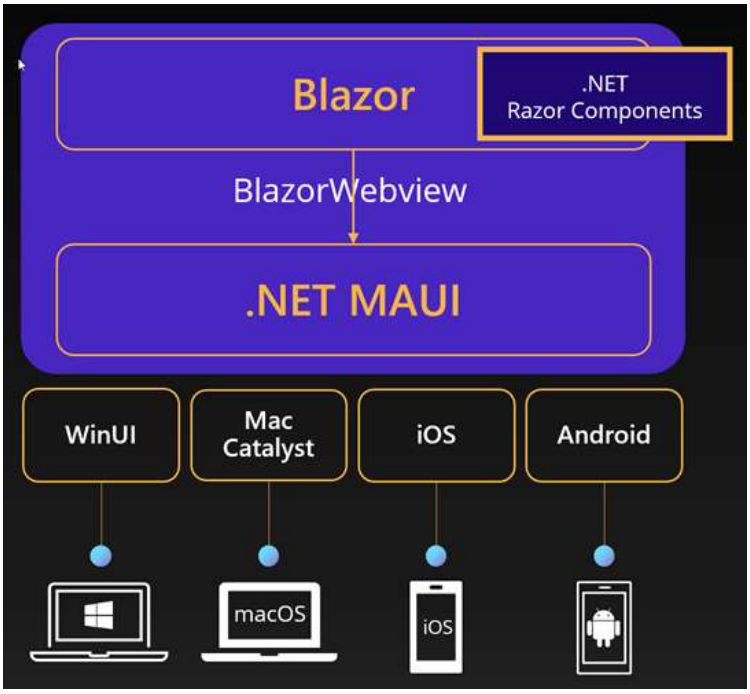
- File system
- Sensors
- OS APIs
- Native controls

Key Advantages

- Reuse Blazor UI across desktop & mobile
- No WebAssembly limitations
- Native performance
- Full device access

Typical Use Cases

- Enterprise desktop applications
- Mobile apps with shared web UI
- Line-of-business tools



---

8. Summary – Choosing the Right Hosting Model

Scenario	Recommended Model
Internal enterprise apps	Blazor Server
Public SPA with scale	Blazor WebAssembly
Offline-first apps	Blazor WebAssembly
Desktop & mobile apps	Blazor Hybrid
Real-time collaboration	Blazor Server

Final Note

Blazor hosting models are **not competitors**, but **strategic options**.
Modern architectures often **combine models**:

- Blazor WASM frontend
- ASP.NET Core Web API backend
- SignalR for selective real-time features
- Blazor Hybrid for companion desktop/mobile apps