# Blazor — Apunte de teoría (Módulo 1)

Basado en la transcripción del primer módulo del curso (Felipe Gavilán).

## 1) Panorama general

* **Blazor** es un framework de .NET para construir **aplicaciones web interactivas** que se usan desde un **navegador**.
* Gracias a **WebAssembly (WASM)**, podemos ejecutar **C#** en el navegador (además de JavaScript).
* Se programa con un **paradigma de componentes** y se puede **interoperar con JavaScript** (llamar JS desde C# y viceversa).

**Ventajas clave de usar .NET/C# en el navegador** 1. Aprovechar **ecosistema .NET**. 2. Usar **C#**, con **LINQ** y **asincronía** (async/await). 3. **Compartir código** entre *front‑end* y *back‑end* (ASP.NET Core). 4. **Componentización** (reutilización, composición, testeo más simple).

## 2) Modelos de alojamiento de Blazor

Blazor define **cómo** y **dónde** corre tu app.

### 2.1 Blazor WebAssembly (lado del **cliente**)

* El navegador **descarga** un *runtime* especializado de .NET y las **DLL** de la app; todo se ejecuta **localmente** en el navegador.
* Se puede crear **solo front‑end** (app estática que consume cualquier Web API: .NET, Node, Go, etc.) o **ASP.NET Core hosted** (solución con **proyecto cliente + servidor** Web API, ideal para desplegar juntos y **compartir código**).
* **Compatibilidad:** requiere navegadores con **WebAssembly** (la gran mayoría modernos, incluso móviles). No IE antiguos.
* **Escalabilidad:** muy alta (el trabajo corre en el cliente, menos carga en el servidor).
* **Costo inicial:** primera carga más pesada por descarga del runtime (se **cachea** después).

### 2.2 Blazor Server (lado del **servidor**)

* La app **corre en el servidor**; el cliente se comunica por **SignalR** (tiempo real).
* **Ventajas:** sin descarga de runtime en el cliente → **arranque más rápido** y funciona en **dispositivos limitados**.
* **Contras:**
  + Requiere **servidor activo** permanente (sin server, la app no funciona).
  + **Latencia** perceptible si la conexión del cliente es mala (cada interacción viaja al servidor).
  + **Escalabilidad** desafiante (múltiples usuarios comparten recursos del servidor).
* **Compatibilidad:** prácticamente cualquier navegador moderno. IE11 solo con **polyfills** (según docs; versiones previas de IE no soportan).

### 2.3 Blazor Híbrido (desktop y móviles)

* App **nativa** (WinForms, WPF, **.NET MAUI**) que hospeda una **WebView** para renderizar componentes Blazor.
* **Ventajas:**
  + **Código compartido** entre web, escritorio y móvil.
  + **Acceso total a capacidades nativas** del dispositivo.
  + **Rendimiento local** (como WebAssembly, ejecuta en el dispositivo).
* **Contras:** despliegue como **app nativa** (p. ej., publicar en Play Store/App Store). Más fricción para el usuario que una URL.

## 3) Dónde se puede usar cada modelo

* **Blazor WebAssembly:** navegadores con soporte **WASM**. (Cobertura global ~muy alta según *Can I use*; en la transcripción se menciona ~96%).
* **Blazor Server:** navegadores modernos; IE11 con **polyfills**.
* **Blazor Híbrido:** apps **nativas** desktop/móvil con WinForms, WPF o MAUI.

## 4) Comparativa rápida (resumen)

| Característica | WebAssembly (Cliente) | Server (Servidor) | Híbrido (Nativo) |
| --- | --- | --- | --- |
| **Lenguaje** en UI | C# (con interoperabilidad JS) | C# (interoperabilidad JS) | C# (interoperabilidad JS) |
| **Descarga inicial** | **Alta** (runtime + DLL) → cachea | **Baja** | **N/A** (instalada) |
| **Dispositivos limitados** | Puede costar más | **Mejor** (procesa el servidor) | Similar a WASM (ejecuta local) |
| **Latencia UI** | **Baja** (local) | **Puede haber** (SignalR) | **Baja** (local) |
| **Server activo** | **No** (puede servir estático/CDN) | **Sí, siempre** | **No** para ejecutar (sí para backend si aplica) |
| **ASP.NET Core requerido** | **No obligatorio** | **Sí** | **No obligatorio** |
| **Depende de WASM** | **Sí** | **No** | **No** |
| **Escalabilidad** | **Muy alta** | **Retos** | **Muy alta** |
| **CDN/archivos estáticos** | **Sí** (hosting barato/gratis) | **No** | **No aplica** |
| **Modo offline** | **Posible** (PWA y cache) | **No** | **Sí** (app nativa) |
| **Acceso nativo** | **Limitado** (sandbox navegador) | **Limitado** | **Completo** |

**Elección del modelo**: depende de requisitos de compatibilidad, latencia, escalabilidad, acceso a capacidades nativas y estrategia de despliegue/costos.

## 5) Preparar el entorno de desarrollo

### 5.1 Visual Studio (Windows)

1. Descargar **Visual Studio** (edición **Community**) desde el sitio oficial.
2. En el instalador, activar la carga de trabajo **“ASP.NET and web development”** (y otras si las necesitas).
3. Verificar .NET: abrir cmd y ejecutar dotnet --version (en la transcripción se usa .NET 7).
4. Crear proyecto: **Create a new project → Blazor WebAssembly App** (no la plantilla “Empty”).
   * **.NET**: seleccionar la versión objetivo.
   * **ASP.NET Core Hosted**: **marcar** si quieres solución **cliente+servidor** (Web API para BD).
   * HTTPS: opcional según curso; en la transcripción se deja sin marcar.
5. Ejecutar: **Ctrl+F5**.

### 5.2 Visual Studio Code (Windows/macOS/Linux)

1. Instalar **VS Code**.
2. Instalar **.NET SDK** desde dotnet.microsoft.com.
3. Extensiones recomendadas: **C#** (y otras de .NET).
4. Verificar .NET: en terminal de la carpeta de tu proyecto dotnet --version.
5. Ver plantillas: dotnet new list.
6. Crear Blazor WebAssembly **hosted**:

* dotnet new blazorwasm --hosted -n BlazorPeliculas -o BlazorPeliculas

1. Abrir en VS Code: code .
2. Ejecutar (proyecto **Server**):

* cd BlazorPeliculas/Server  
  dotnet watch run
* Si VS Code pregunta por *assets* de compilación, **aceptar** y seleccionar el proyecto **Server** como *launch*.

## 6) Interoperabilidad y arquitectura

* **Interoperabilidad JS**: desde Blazor puedes **invocar funciones JS**, y desde JS puedes **invocar métodos .NET**.
* **Compartir código**: en la solución *hosted* (Cliente + Server), es común agregar un proyecto **Compartido** (contratos/DTOs/validaciones) para **evitar duplicación** entre front‑end y back‑end.

## 7) Puntos a memorizar (tipo examen)

* **Definición de Blazor y rol de WASM**.
* **Ventajas** de C# en el navegador (ecosistema, LINQ, async/await, compartir código, componentes).
* **Diferencias** entre **WebAssembly vs Server vs Híbrido** (ejecución, latencia, escalabilidad, offline, acceso nativo, requisitos).
* **Cuándo** conviene **ASP.NET Core Hosted** (deploy conjunto, compartir código, Web API integrado).
* **Compatibilidad**: WASM requiere navegadores modernos; Blazor Server funciona en casi todos (IE11 con polyfills); Híbrido = apps nativas (WinForms/WPF/MAUI).
* **Despliegue**: WASM puede servirse como **estático/CDN**; Server necesita **servidor**; Híbrido requiere **tienda/app installer**.

## 8) Mini‑glosario

* **WASM / WebAssembly**: formato binario portable para ejecutar código de alto rendimiento en el navegador.
* **SignalR**: biblioteca para **comunicación en tiempo real** (WebSockets/Long Polling) usada por Blazor Server.
* **CDN**: red de distribución de contenido para servir **archivos estáticos** cerca del usuario.
* **Polyfill**: script que agrega soporte de características modernas en navegadores antiguos.

## 9) Comandos útiles (CLI)

# Ver versión .NET  
dotnet --version  
  
# Listar plantillas disponibles  
dotnet new list  
  
# Crear Blazor WebAssembly “hosted” (cliente + servidor)  
dotnet new blazorwasm --hosted -n MiApp -o MiApp  
  
# Ejecutar con recarga en caliente (proyecto Server)  
cd MiApp/Server  
dotnet watch run

## 10) Checklist de proyecto inicial (WASM hosted)

* Plantilla **Blazor WebAssembly** con opción **ASP.NET Core Hosted**.
* Proyecto **Client**: componentes, páginas, servicios HTTP.
* Proyecto **Server**: ASP.NET Core Web API (acceso a **BD**).
* (Opcional) Proyecto **Shared**: modelos/DTOs compartidos.
* Configurar **launch** en VS Code: lanzar **Server**.
* Probar navegación y peticiones **HTTP** desde Client a Server.

### Nota final

Este apunte resume el **Módulo 1**: conceptos, ventajas, modelos de alojamiento, compatibilidad, entorno y primeros pasos, siguiendo la transcripción. Lo que se aprende en **WASM** aplica en gran medida a **Server** e **Híbrido** (con diferencias puntuales).