Integración de API con Front-end en Blazor

Curso .NET – IT School

IT SCHOOL
LA POSTA!

¿Qué es Blazor?

- Blazor es un framework de desarrollo web creado por Microsoft que permite crear aplicaciones web interactivas del lado del cliente utilizando C# y .NET en lugar de JavaScript. Blazor se basa en el modelo de componentes, lo que permite una programación más modular y reutilizable. Es parte del ecosistema de ASP.NET Core.

Características Clave

- **Componentes Reutilizables:** Utiliza componentes que se pueden reutilizar en diferentes partes de la aplicación.
- **Data Binding:** Soporta data binding bidireccional, lo que facilita la sincronización de datos entre la interfaz de usuario y el modelo.
- **Rendimiento:** Ofrece un rendimiento comparable a las aplicaciones desarrolladas con otras tecnologías de front-end.
- **Compatibilidad:** Se integra fácilmente con el ecosistema de ASP.NET Core, permitiendo el uso de librerías y herramientas existentes.

Historia y Evolución

- **Inicios (2018):** Blazor fue anunciado por primera vez en 2018 como un proyecto experimental de Microsoft, con la idea de traer las capacidades de .NET al navegador utilizando WebAssembly.
- **Blazor Server (2019):** Con la liberación de .NET Core 3.0 en 2019, Blazor Server se lanzó como una opción de producción estable. En esta versión, la lógica de la aplicación se ejecuta en el servidor y se comunica con el navegador a través de una conexión SignalR.
- **Blazor WebAssembly (2020):** En 2020, se lanzó Blazor WebAssembly, permitiendo que el código C# se ejecute directamente en el navegador utilizando WebAssembly, sin necesidad de una conexión constante al servidor.
- **Blazor en .NET 5 y .NET 6:** Con la llegada de .NET 5 y .NET 6, Blazor continuó mejorando en términos de rendimiento, herramientas de desarrollo y características adicionales como la pre-renderización y el soporte de PWA (Progressive Web Apps).

Blazor en .NET 8

- **Blazor Web App:**
  - La plantilla de Blazor Web App ahora ofrece un punto de inicio unificado para desarrolladores, permitiendo combinar la renderización del lado del servidor y del cliente en una sola solución. Esto facilita el desarrollo de aplicaciones con renderización estática rápida desde el servidor y una interactividad completa del cliente, mejorando tanto la velocidad de carga inicial como la experiencia del usuario.
  - Se ha mejorado el soporte para la renderización interactiva del lado del servidor (SSR), permitiendo que las interacciones de la interfaz de usuario sean manejadas en tiempo real sin necesidad de crear endpoints API para acceder a los recursos del servidor.
  - La nueva plantilla de Blazor Web App incluye opciones convenientes para configurar la autenticación y elegir modos de renderización interactivos (Servidor, WebAssembly, o ambos con Auto). Además, el nuevo Blazor scaffolder permite generar rápidamente páginas CRUD utilizando Entity Framework Core y el componente QuickGrid.

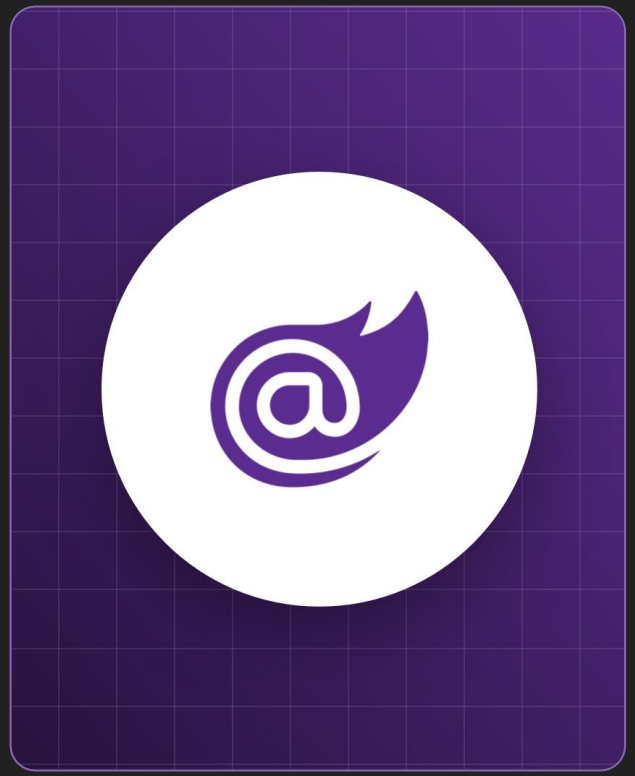
Blazor

- Blazor se destaca por permitir a los desarrolladores .NET utilizar sus habilidades existentes en el desarrollo de aplicaciones web, eliminando la necesidad de aprender un nuevo lenguaje como JavaScript. Además, la integración con el ecosistema de ASP.NET Core facilita la creación de aplicaciones full-stack con una base de código unificada.

Característica	Blazor
Lenguaje	C#
Arquitectura	Modelo de Componentes
Data Binding	Bidireccional
Ecosistema	Integrado con .NET

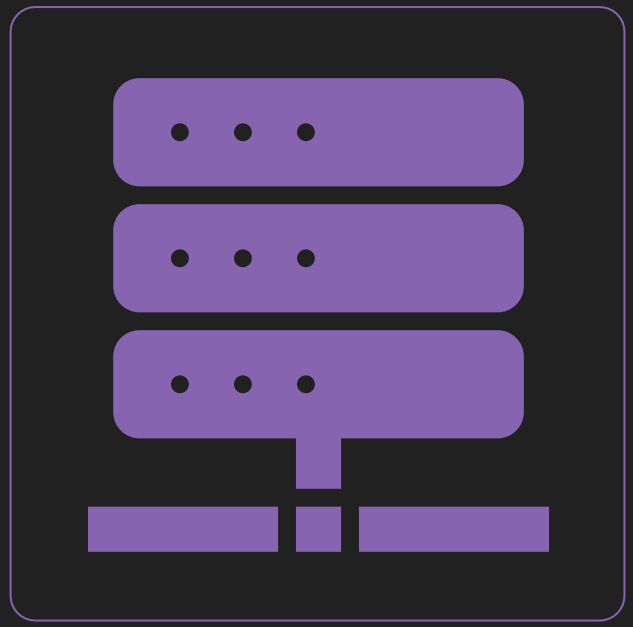
Tipos de Blazor

- Blazor viene en dos sabores principales: Blazor Server y Blazor WebAssembly. Cada uno tiene su propio modelo de operación y casos de uso específicos.



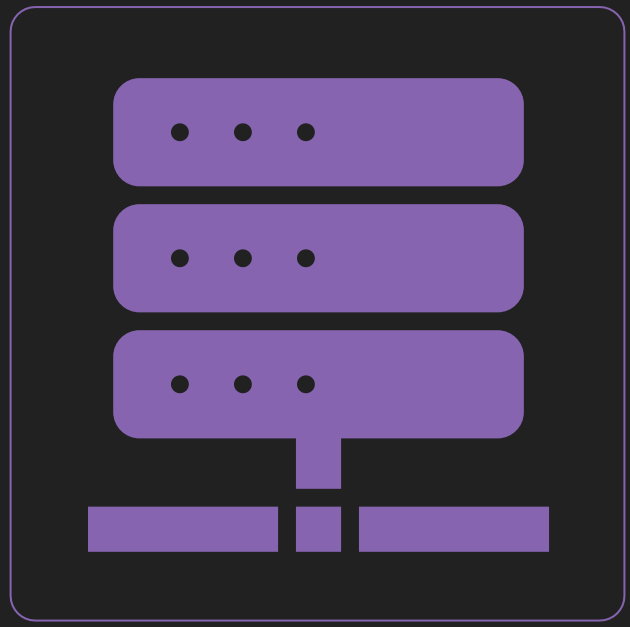
Blazor Server

- Blazor Server ejecuta la lógica de la aplicación en el servidor. Los eventos de la interfaz de usuario, como los clics de botón, se envían al servidor utilizando SignalR, una biblioteca de ASP.NET para la comunicación en tiempo real, y el DOM se actualiza en el navegador en consecuencia.
- El código de la aplicación se ejecuta en el servidor, mientras que una conexión en tiempo real mantiene sincronizada la interfaz de usuario del cliente.



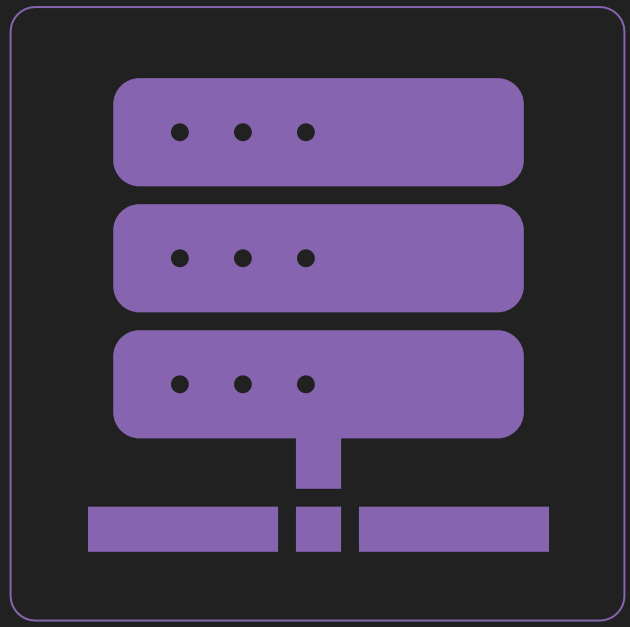
Blazor Server - Características Clave

- **Renderizado rápido:** Debido a que la lógica de la aplicación se ejecuta en el servidor, el tiempo de carga inicial puede ser más rápido.
- **Tamaño reducido del cliente:** La cantidad de datos enviados al cliente es mínima, lo que resulta en tiempos de carga más rápidos.
- **Compatibilidad completa con .NET:** Dado que la aplicación se ejecuta en el servidor, tiene acceso a toda la funcionalidad de .NET, incluidas las bibliotecas que no son compatibles con WebAssembly.



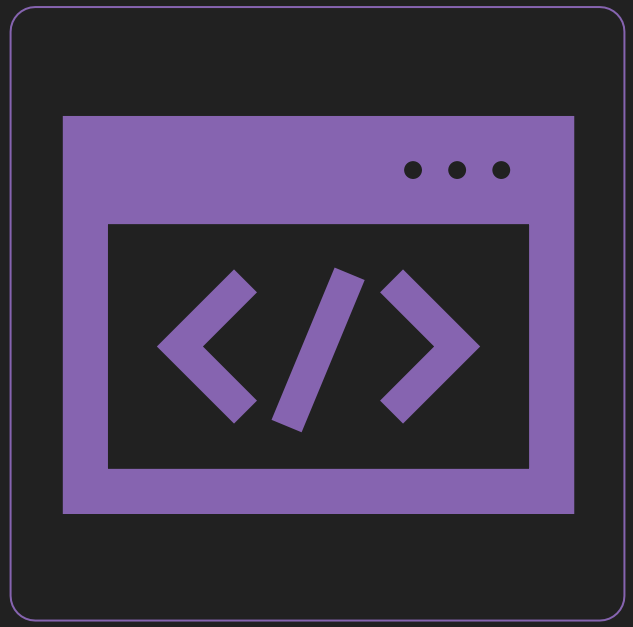
Blazor Server - Casos de Uso

- **Aplicaciones internas:** Donde la latencia de red no es un problema y se requiere acceso completo a recursos del servidor.
- **Aplicaciones de baja latencia:** Donde la conexión de red es rápida y confiable, permitiendo una experiencia de usuario fluida.
- **Aplicaciones con alta seguridad:** Donde la lógica y los datos sensibles se mantienen en el servidor, reduciendo el riesgo de exposición.



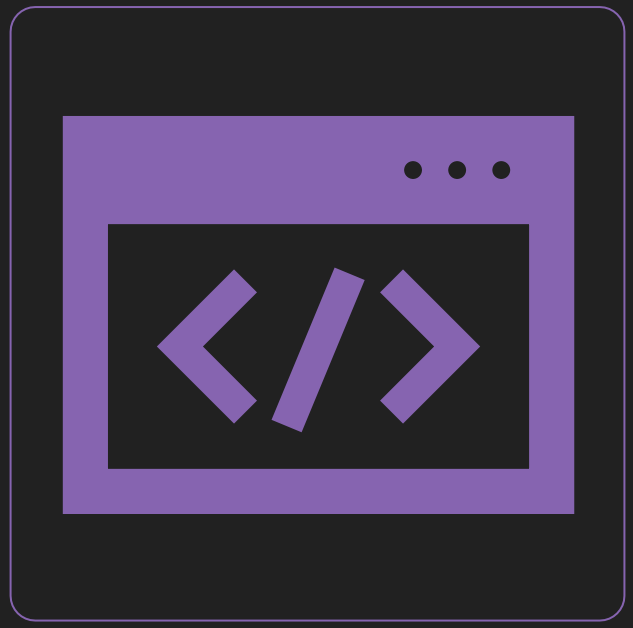
Blazor WebAssembly

- Blazor WebAssembly permite ejecutar la lógica de la aplicación directamente en el navegador del cliente. Utiliza WebAssembly, un formato binario que permite la ejecución de código en casi cualquier navegador moderno a una velocidad cercana a la nativa.
- La aplicación se descarga al cliente y se ejecuta completamente en el navegador sin necesidad de comunicación constante con el servidor.



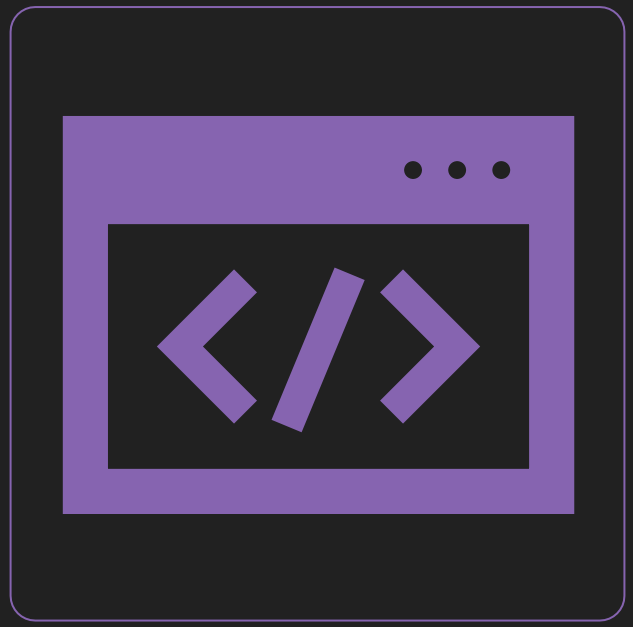
Blazor WebAssembly - Características Clave

- **Desconectado del servidor:** Las aplicaciones pueden funcionar sin una conexión constante al servidor una vez que se han cargado.
- **Escalabilidad:** Dado que la lógica de la aplicación se ejecuta en el cliente, la carga en el servidor es menor.
- **Distribución estática:** Las aplicaciones se pueden alojar en cualquier servidor web o incluso en redes de entrega de contenido (CDN) para una distribución rápida y eficiente.

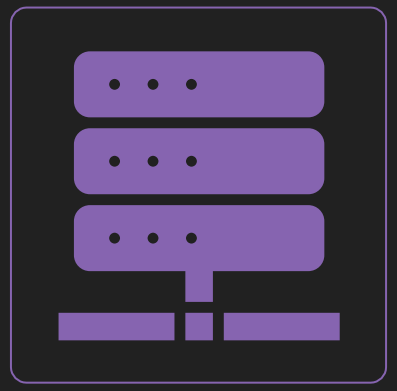


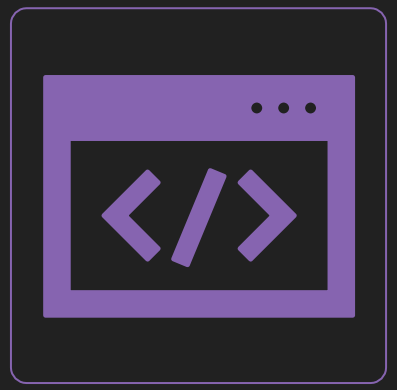
Blazor WebAssembly - Casos de Uso

- **Aplicaciones móviles y de escritorio:** Donde los usuarios necesitan trabajar sin una conexión continua a Internet.
- **Aplicaciones distribuidas globalmente:** Donde la latencia de red puede ser alta y se necesita una carga distribuida para mejorar la experiencia del usuario.
- **Aplicaciones ligeras:** Donde se requiere una interacción rápida y responsiva del usuario sin la latencia de las solicitudes de red.



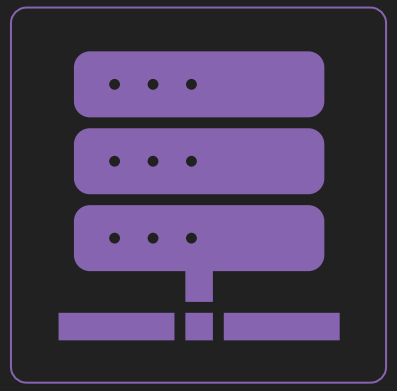
Blazor - Diferencias Clave y Casos de Uso

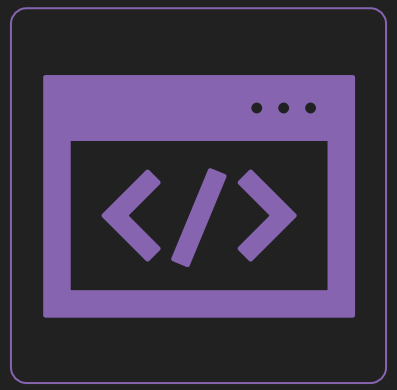




- Rendimiento y Latencia:
  - **Blazor Server:** Adecuado para aplicaciones con baja latencia de red y donde se necesita acceso completo a los recursos del servidor.
  - **Blazor WebAssembly:** Ideal para aplicaciones que requieren alta interactividad y pueden beneficiarse de la ejecución local en el cliente.

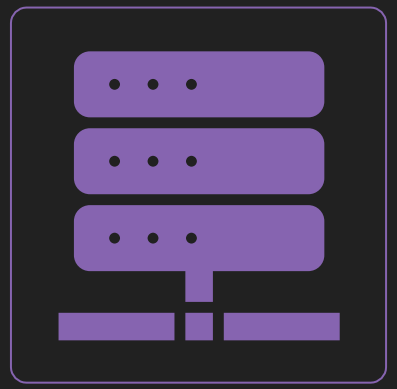
Blazor - Diferencias Clave y Casos de Uso

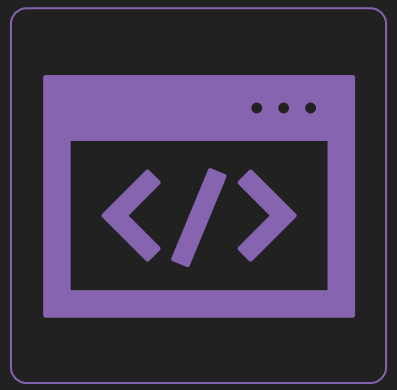




- Carga del Servidor:
  - **Blazor Server:** La lógica de la aplicación se ejecuta en el servidor, lo que puede incrementar la carga en aplicaciones con muchos usuarios concurrentes.
  - **Blazor WebAssembly:** La lógica se ejecuta en el cliente, distribuyendo la carga y reduciendo el impacto en el servidor.

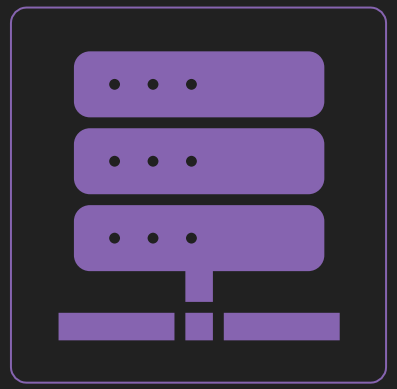
Blazor - Diferencias Clave y Casos de Uso

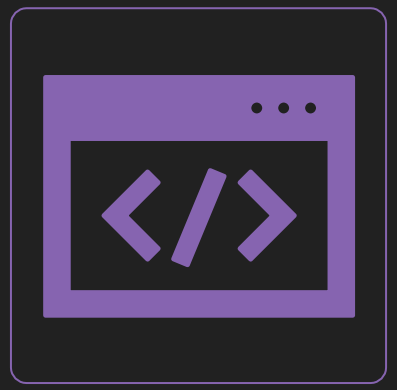




- Seguridad:
  - **Blazor Server:** La lógica y los datos sensibles permanecen en el servidor, proporcionando una capa adicional de seguridad.
  - **Blazor WebAssembly:** Los datos y la lógica se descargan al cliente, lo que puede requerir consideraciones adicionales de seguridad.

Blazor - Diferencias Clave y Casos de Uso





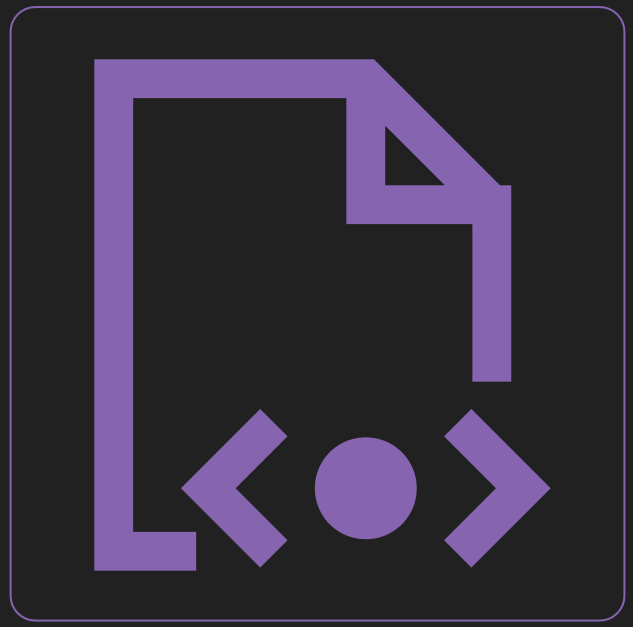
- Desarrollo y Herramientas:
  - **Blazor Server:** Ofrece una experiencia de desarrollo completa con acceso a todas las capacidades de .NET y herramientas de desarrollo en el servidor.
  - **Blazor WebAssembly:** Las aplicaciones pueden beneficiarse de características avanzadas de .NET, pero están limitadas por las capacidades de WebAssembly y la necesidad de optimizar el tamaño del paquete para una carga rápida.

Blazor - Casos de Uso Combinados

- Con las actualizaciones en .NET 8, ahora es posible crear aplicaciones Blazor que combinan ambos modelos de hospedaje, aprovechando lo mejor de cada uno según las necesidades específicas del usuario. Esto permite, por ejemplo, usar Blazor Server para el renderizado inicial rápido y Blazor WebAssembly para interactividad del cliente sin conexión.

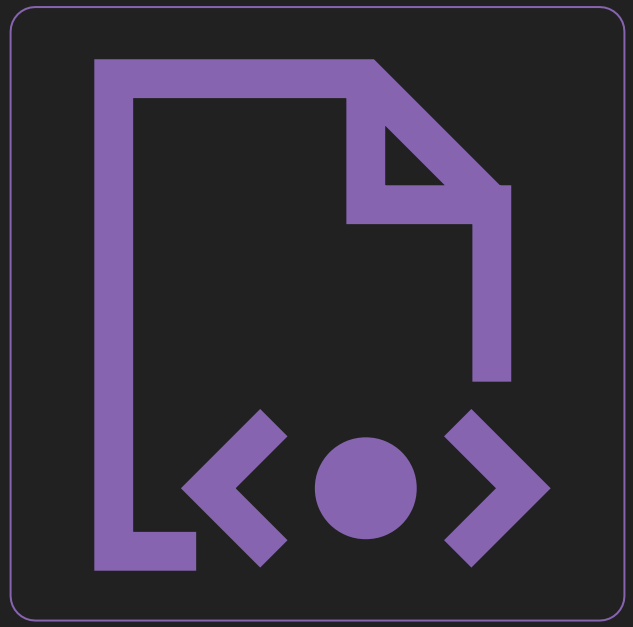
¿Qué son los componentes?

- En Blazor, los componentes son los bloques de construcción fundamentales de las aplicaciones. Cada componente es una combinación de HTML, CSS y C# que puede ser reutilizado en diferentes partes de la aplicación. Los componentes se definen utilizando archivos .razor y pueden anidar otros componentes, manejar eventos y mantener el estado.



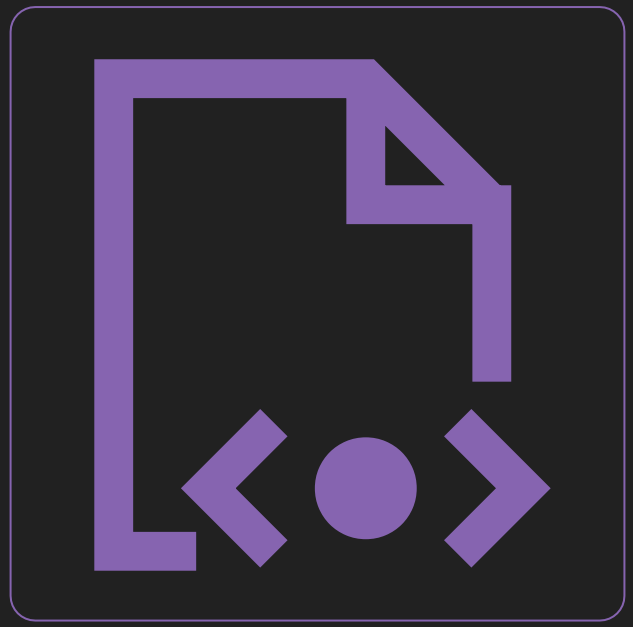
Características de los componentes de Blazor

- **Reutilizables:** Los componentes pueden ser reutilizados en diferentes partes de la aplicación.
- **Anidables:** Un componente puede contener otros componentes.
- **Encapsulados:** Los componentes encapsulan su lógica, lo que facilita el mantenimiento y la prueba.
- **Interactivos:** Pueden manejar eventos del usuario y actualizar la interfaz de usuario de manera dinámica.



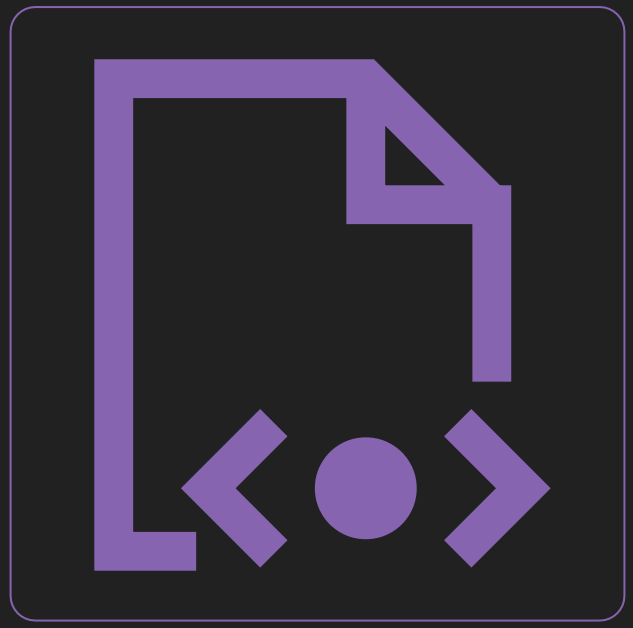
Sintaxis de Razor

- Razor es un motor de plantillas que se utiliza para generar HTML dinámico en las aplicaciones ASP.NET Core. En Blazor, Razor se utiliza para definir la estructura y el comportamiento de los componentes. La sintaxis de Razor permite intercalar C# con HTML de manera fluida.



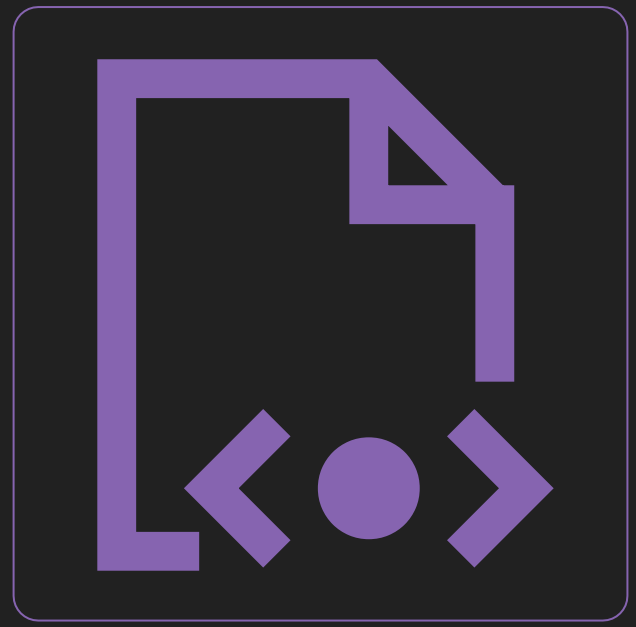
Elementos clave de la sintaxis de Razor

- **Directivas:** Utilizadas para definir componentes, importar namespaces, y otros elementos. Por ejemplo, @page, @using, @inherits.
- **Interpolación:** Permite insertar expresiones C# en el HTML. Se usa @ seguido de la expresión C#, como @DateTime.Now.
- **Bloques de código:** Permiten escribir bloques de código C# dentro del HTML. Se definen con @code { ... }.
- **Bindings:** Facilitan la vinculación de datos entre el modelo y la vista. Se usa @bind para crear enlaces bidireccionales, como <input @bind="property" />.



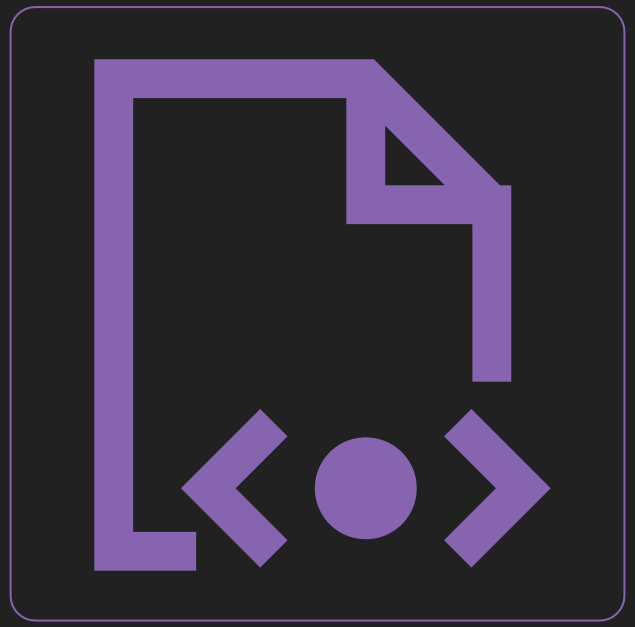
Ejemplo

```
@page "/micomponente"

<h3>Mi Componente</h3>

<p>La hora actual es: @DateTime.Now</p>

<button @onclick="ActualizarHora">Actualizar Hora</button>

@code {
    private void ActualizarHora()
    {
        StateHasChanged();
    }
}
```



Data Binding en Blazor

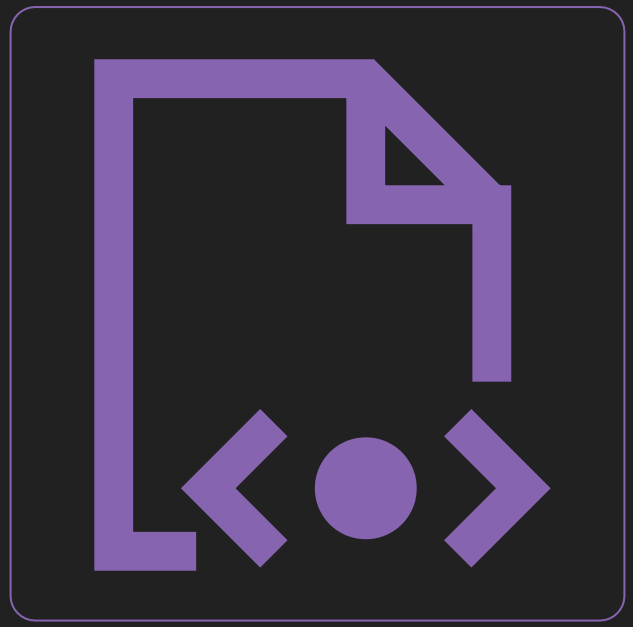
- El data binding en Blazor es una característica poderosa que permite la sincronización automática entre el modelo de datos y la interfaz de usuario. Blazor soporta tanto el binding unidireccional (one-way binding) como el bidireccional (two-way binding).



One-way Binding (Enlace Unidireccional)

- El binding unidireccional se utiliza para mostrar datos en la interfaz de usuario. Los cambios en el modelo de datos se reflejan en la interfaz, pero los cambios en la interfaz no afectan al modelo de datos.

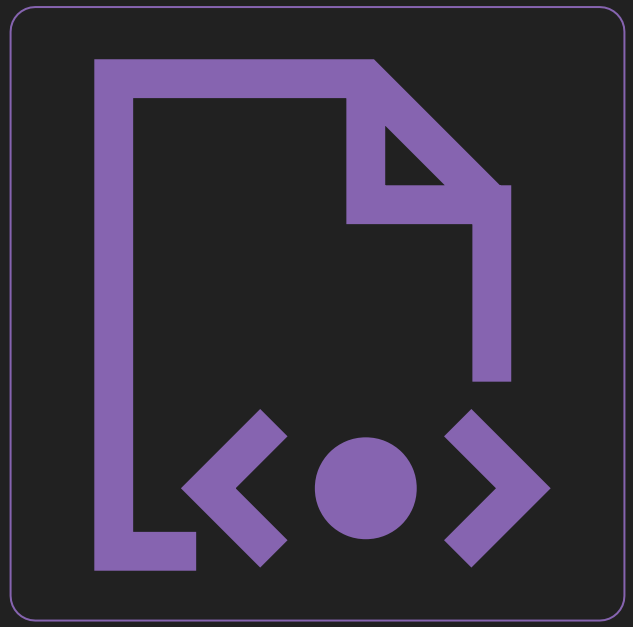
```
<h3>El valor del contador es: @contador</h3>

@code {
    private int contador = 0;
}
```



Two-way Binding (Enlace Bidireccional)

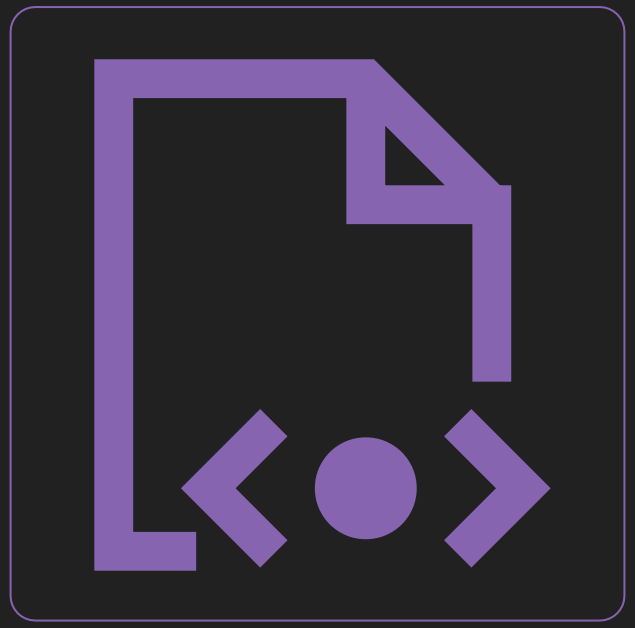
- El binding bidireccional permite que los cambios en el modelo de datos se reflejen en la interfaz de usuario y viceversa. Es especialmente útil para formularios y entradas de usuario.

```
<input @bind="nombre" />
<p>El nombre es: @nombre</p>

@code {
    private string nombre = "Blazor";
}
```



Eventos y Métodos

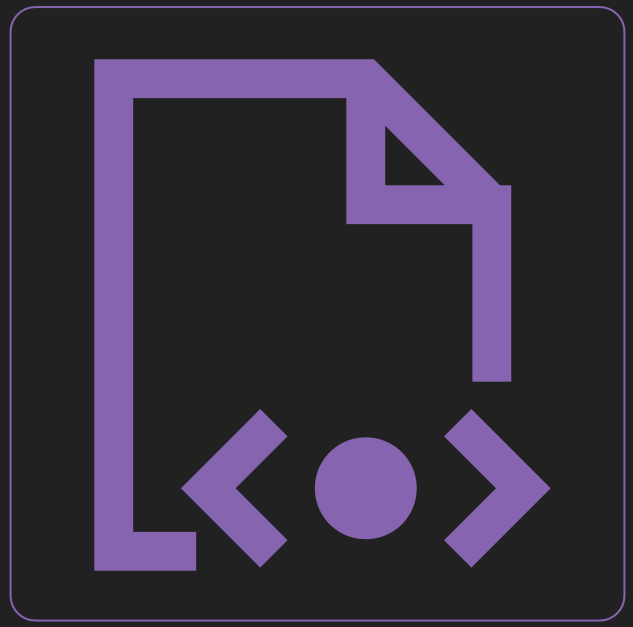
- Blazor permite manejar eventos del DOM y llamar métodos de C# en respuesta a estos eventos. Esto se hace utilizando atributos de evento como @onclick, @onchange, etc.



Manejo de Eventos

- Para manejar un evento, se define un método en el código del componente y se vincula al evento en la interfaz de usuario.

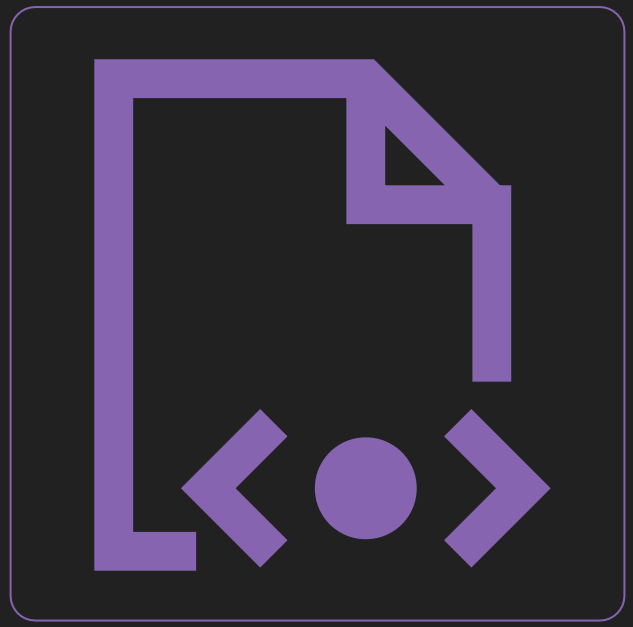
```
<button @onclick="IncrementarContador">Incrementar</button>
<p>El valor del contador es: @contador</p>

@code {
    private int contador = 0;

    private void IncrementarContador()
    {
        contador++;
    }
}
```



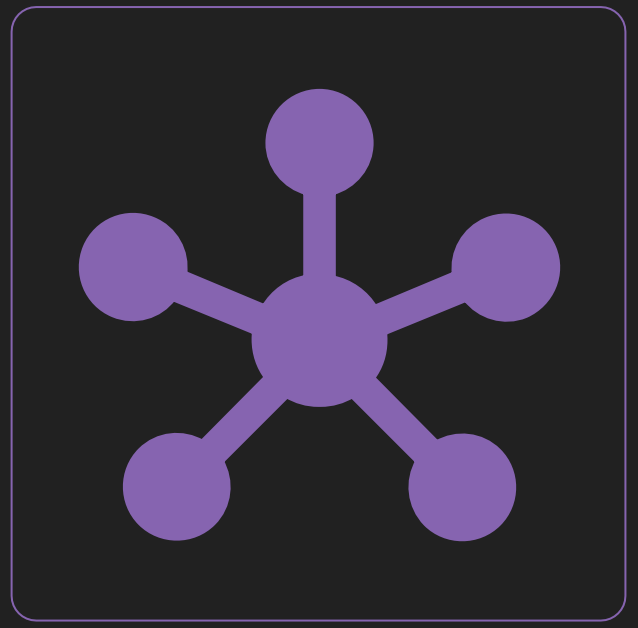
Eventos y Pasar Parámetros

- También es posible pasar parámetros a los métodos de manejo de eventos.

```
<button @onclick="() => IncrementarContadorConValor(5)">Incrementar en 5</button>
<p>El valor del contador es: @contador</p>

@code {
    private int contador = 0;

    private void IncrementarContadorConValor(int valor)
    {
        contador += valor;
    }
}
```

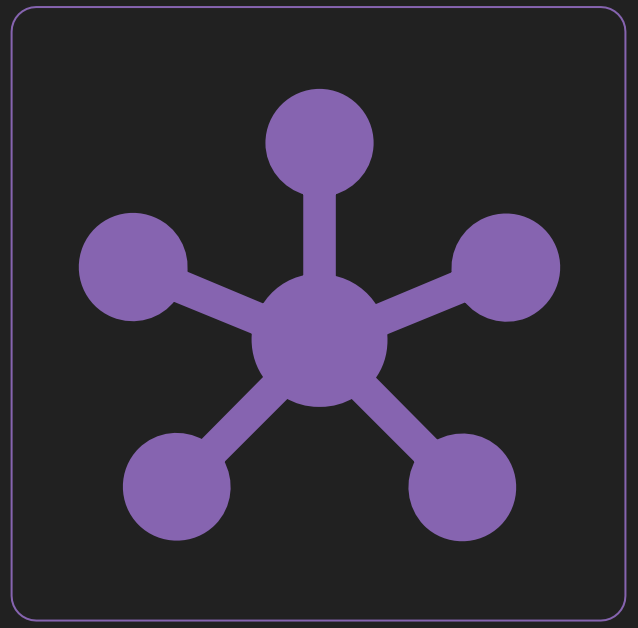


HTTPClient y Blazor



- Blazor facilita la interacción con servicios web externos mediante el uso de HttpClient. Este componente permite realizar solicitudes HTTP para obtener, enviar, actualizar y eliminar datos. Es una herramienta esencial para la integración de APIs en aplicaciones Blazor.

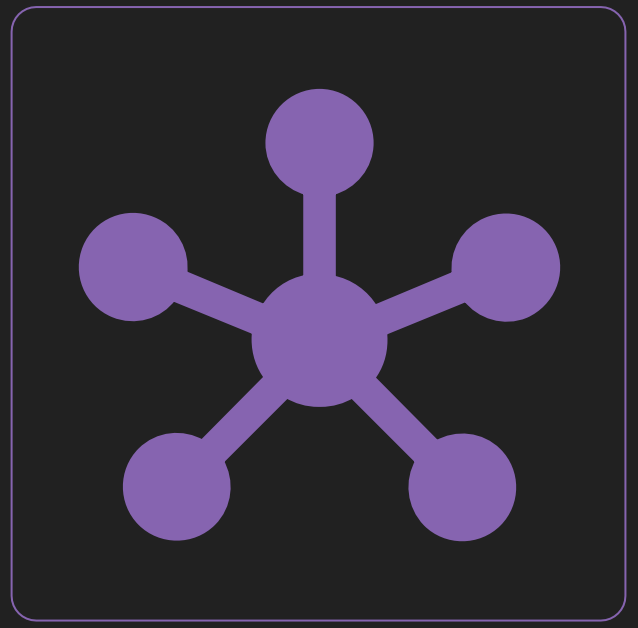
Configuración del HttpClient



- **Inyección de Dependencias:** En Blazor, HttpClient se configura y se inyecta utilizando el contenedor de dependencias de ASP.NET Core. Dependiendo del tipo de proyecto (Blazor WebAssembly o Blazor Server), la configuración puede variar ligeramente.

```
@inject HttpClient Http
```

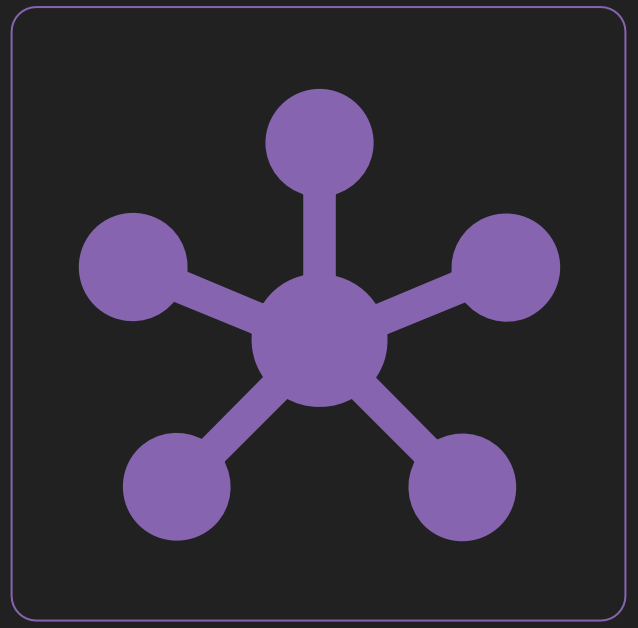
Configuración del HttpClient



- **Blazor WebAssembly:** En Program.cs, configura HttpClient para que apunte a la base de la URL de la API.

```
using System.Net.Http;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Microsoft.Extensions.DependencyInjection;

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

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

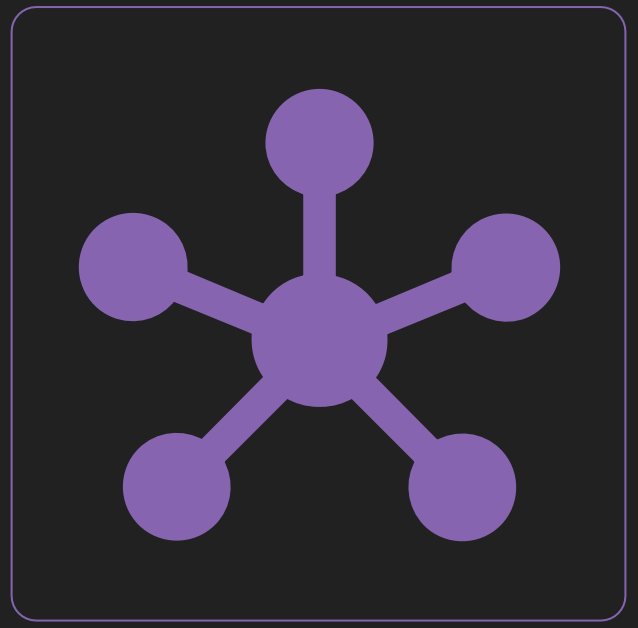
Configuración del HttpClient



- **Blazor Server:** En Program.cs, configura HttpClient utilizando HttpClientFactory.

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddHttpClient();

var app = builder.Build();
app.Run();
```

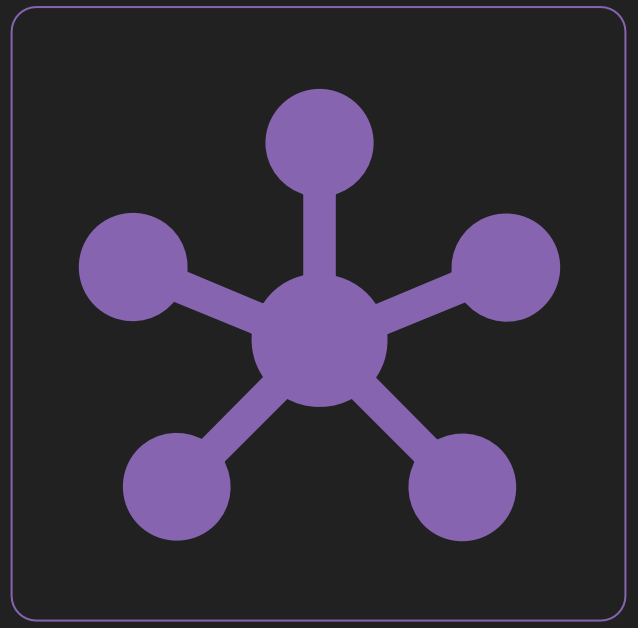
Realizar Solicitudes HTTP



- **Solicitud GET:** Para realizar una solicitud GET y obtener datos de una API, puedes utilizar el método GetAsync.

```
@code {
    private WeatherForecast[] forecasts;

    protected override async Task OnInitializedAsync()
    {
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("WeatherForecast");
    }

    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureC { get; set; }
        public string Summary { get; set; }
    }
}
```

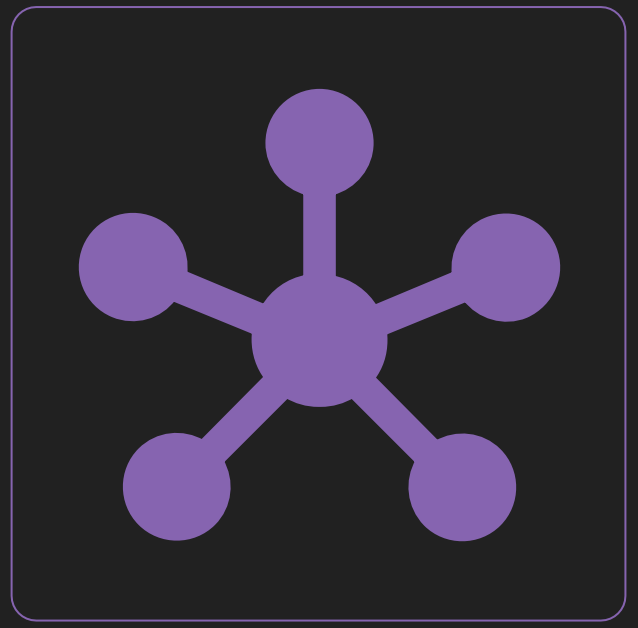
Realizar Solicitudes HTTP



- **Solicitud POST:** Para enviar datos a una API, utiliza el método PostAsync.

```
@code {
    private NewDataModel newData = new NewDataModel();

    private async Task SendData()
    {
        var response = await Http.PostAsJsonAsync("api/data", newData);
        if (response.IsSuccessStatusCode)
        {
            // Manejar el caso de éxito
        }
    }

    public class NewDataModel
    {
        public string Name { get; set; }
    }
}
```

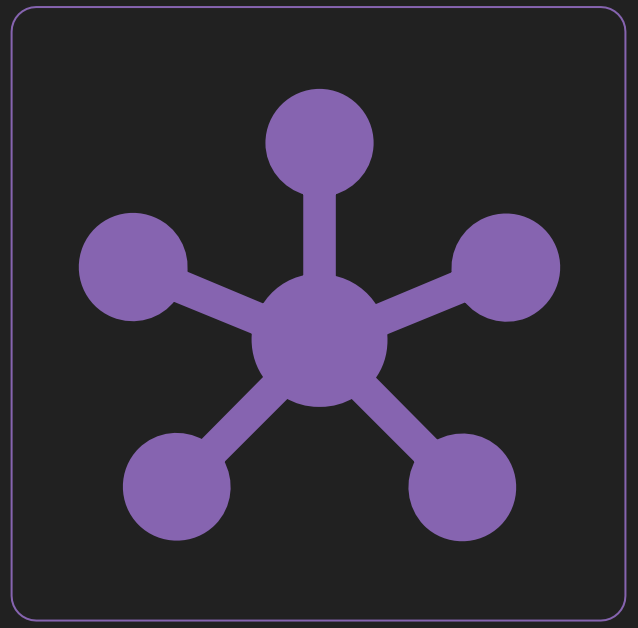
Realizar Solicitudes HTTP



- **Solicitud PUT**: Para actualizar datos en una API, utiliza el método PutAsync.

```
@code {
    private UpdateDataModel updateData = new UpdateDataModel();

    private async Task UpdateData()
    {
        var response = await Http.PutAsJsonAsync("api/data/1", updateData);
        if (response.IsSuccessStatusCode)
        {
            // Manejar el caso de éxito
        }
    }

    public class UpdateDataModel
    {
        public string Name { get; set; }
    }
}
```

Realizar Solicitudes HTTP



- **Solicitud DELETE:** Para eliminar datos de una API, utiliza el método DeleteAsync.

```
@code {
    private async Task DeleteData()
    {
        var response = await Http.DeleteAsync("api/data/1");
        if (response.IsSuccessStatusCode)
        {
            // Manejar el caso de éxito         }
    }
}
```

Consumo de API Protegida por JWT en Blazor

- Consumir una API protegida por JWT (JSON Web Token) en Blazor implica varios pasos, que incluyen la autenticación del usuario, la obtención del token JWT y el uso de este token en las solicitudes HTTP para acceder a los recursos protegidos.

Consumo de API Protegida por JWT en Blazor

- **Configuración de la API:** La configuración de la API para que use JWT lo hicimos en el módulo anterior. Lo que nos resta hacer es configurar Blazor para que la consuma.
- Este ejemplo vamos a hacerlo con Blazor WebAssembly, la API puede ejecutarse en el mismo servidor de nuestra Blazor Web App o un proyecto de ASP.NET Core dentro de la misma solución. En ese caso "BaseAddress" tiene que apuntar a la ruta base donde se esté ejecutando la API.

Consumo de API Protegida por JWT en Blazor

- **Configuración de HttpClient:** Configura HttpClient en Program.cs para incluir el token JWT en las solicitudes HTTP. A su vez creamos TokenProvider.cs para manejar los tokens.

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

builder.Services.AddScoped<TokenProvider>();
```

Consumo de API Protegida por JWT en Blazor

```
public class TokenProvider
{
    private readonly HttpClient _httpClient;

    public TokenProvider(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public async Task<string> GetTokenAsync(string username, string password)
    {
        var response = await _httpClient.PostAsJsonAsync("api/auth/login", new { username, password });
        response.EnsureSuccessStatusCode();

        var result = await response.Content.ReadFromJsonAsync<LoginResult>();
        return result.Token;
    }
}

public class LoginResult
{
    public string Token { get; set; }
}
```

Consumo de API Protegida por JWT en Blazor

- **Almacenar el Token:** Usa LocalStorage para almacenar el token JWT en el cliente. Instalamos el paquete Blazored.LocalStorage y luego editamos Program.cs

```
dotnet add package Blazored.LocalStorage
```

```
builder.Services.AddBlazoredLocalStorage();
```

Consumo de API Protegida por JWT en Blazor

- **Crear componente de login:** Creamos un componente para iniciar sesión y así obtener el token.

```
@page "/login"
@inject HttpClient Http
@inject TokenProvider TokenProvider
@inject ILocalStorageService LocalStorage
@inject NavigationManager Navigation

<h3>Login</h3>

<input @bind="username" placeholder="Username" />
<input type="password" @bind="password" placeholder="Password" />
<button @onclick="LoginUser">Login</button>
```

Consumo de API Protegida por JWT en Blazor

```
@code {
    private string username;
    private string password;

    private async Task LoginUser()
    {
        var token = await TokenProvider.GetTokenAsync(username,
password);
        await LocalStorage.SetItemAsync("authToken", token);
        Navigation.NavigateTo("/");
    }
}
```

Consumo de API Protegida por JWT en Blazor

- **Incluir el Token en las Solicitudes HTTP:** Configura HttpClient para incluir el token JWT en las solicitudes.

```
public class AuthorizedHttpClient
{
    private readonly HttpClient _httpClient;
    private readonly ILocalStorageService _localStorage;

    public AuthorizedHttpClient(HttpClient httpClient, ILocalStorageService
localStorage)
    {
        _httpClient = httpClient;
        _localStorage = localStorage;
    }
```

Consumo de API Protegida por JWT en Blazor

```
    private async Task AddAuthorizationHeader()
     {
         var token = await
_localStorage.GetItemAsync<string>("authToken");
         if (!string.IsNullOrEmpty(token))
         {
             _httpClient.DefaultRequestHeaders.Authorization =
new AuthenticationHeaderValue("Bearer", token);
         }
     }
```

Consumo de API Protegida por JWT en Blazor

```
    public async Task<T> GetAsync<T>(string uri)
    {
        await AddAuthorizationHeader();
        return await _httpClient.GetFromJsonAsync<T>(uri);
    }

    public async Task<HttpResponseMessage> PostAsync<T>(string uri, T item)
    {
        await AddAuthorizationHeader();
        return await _httpClient.PostAsJsonAsync(uri, item);
    }

    public async Task<HttpResponseMessage> PutAsync<T>(string uri, T item)
    {
        await AddAuthorizationHeader();
        return await _httpClient.PutAsJsonAsync(uri, item);
    }

    public async Task<HttpResponseMessage> DeleteAsync(string uri)
    {
        await AddAuthorizationHeader();
        return await _httpClient.DeleteAsync(uri);
    }
}
```

Consumo de API Protegida por JWT en Blazor

- **Agregar nuevo servicio:** Modifica Program.cs para agregar a nuestro contenedor de dependencias el nuevo servicio creado que agrega el JWT en las solicitudes.

```
builder.Services.AddScoped<AuthorizedHttpClient>();
```

Consumo de API Protegida por JWT en Blazor

- **Consumo de la API en un Componente:** Por último, un ejemplo de cómo usar nuestro nuevo servicio en un componente.

```
@page "/fetchdata"
@inject AuthorizedHttpClient AuthorizedHttp

<h3>Weather Forecast</h3>

@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
```

Consumo de API Protegida por JWT en Blazor

```
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Date</th>
                <th>Temperature (C)</th>
                <th>Summary</th>
            </tr>
        </thead>
        <tbody>
        @foreach (var forecast in forecasts)
        {
            <tr>
                <td>@forecast.Date.ToShortDateString()</td>
                <td>@forecast.TemperatureC</td>
                <td>@forecast.Summary</td>
            </tr>
        }
        </tbody>
    </table>
}
```

Consumo de API Protegida por JWT en Blazor

```
@code {
    private WeatherForecast[] forecasts;

    protected override async Task OnInitializedAsync()
    {
        forecasts = await
AuthorizedHttp.GetAsync<WeatherForecast[]>("WeatherForecast");
    }

    public class WeatherForecast
    {
        public DateTime Date { get; set; }
        public int TemperatureC { get; set; }
        public string Summary { get; set; }
    }
}
```

Material de estudio

- Vídeo oficial del curso:
  - Integración de API con Frontend en Blazor - IT School (youtube.com)
- Vídeos tutoriales:
  - Blazor en 10 minutos - ¿Qué es Blazor? - Aprende Blazor desde Cero (youtube.com)
  - Blazor - ¿Qué es NUEVO en .NET 8? - NOVEDADES (youtube.com)
  - HOLA MUNDO en Blazor 🚀 // Blazor Server Side vs Blazor WebAssembly // (¿Para qué usar Blazor?) (youtube.com)
  - Crear CRUD en Blazor conectado a MVC Api y Sql Server .Net Core (youtube.com)
- Documentación y cursos oficiales:
  - Blazor de ASP.NET Core | Microsoft Learn
  - Ruta de aprendizaje de creación de aplicaciones web con Blazor - Training | Microsoft Learn