

Blazor

Contents

Blazor	7
Ventajas	7
Tipos.....	7
WebAssembly (cliente)	7
Client-side	7
Client-side + ASP.NET Core	7
Server-side	7
Híbrido	8
Modelos de alojamiento	8
Dónde se puede usar	9
WebAssembly	9
Server-side	9
Híbrido	9
Nueva app	10
Razor	11
@page	11
Expresiones implícitas.....	12
Expresiones explícitas	12
Clases	14
_Imports.razor	15
Proyectos	16
foreach	16
For	17
Condicionales	18
Espera.....	19
Comentarios.....	23
MarkupString	23
Componentes.....	26
Routeables	26
Non-Routeables	26

Nuevo componente	26
Parámetros.....	28
Obligatorios.....	29
Arbitrarios	29
Parámetros por defecto.....	30
Eventos.....	31
Data Binding.....	36
Bind Event	36
Bind Get y Bind Set.....	37
Bind After	38
EventCallback.....	38
RenderFragment	42
RenderFragment genérico	44
Ciclo de vida de un componente	46
OnInitialized / OnInitializedAsync.....	46
OnParametersSet / OnParametersSetAsync	46
OnAfterRender / OnAfterRenderAsync	46
ShouldRender.....	46
Inyección de dependencias.....	48
httpClient	49
IJSRuntime	49
Navigation Manager.....	49
Tiempo de vida de servicios	49
Scoped.....	49
Singleton	49
Transient	50
Servicios con interfaces.....	52
Cabecera del HTML	56
PageTitle.....	56
HeadContent.....	57
Clases parciales	58
Layout.....	59
Componente a renderizar	59

Definición del layout	60
Invoke Javascript desde C#	61
Invoke C# desde Javascript	65
Función estática	65
Método de instancia desde JS.....	67
Aislamiento de JS	69
Aislamiento de CSS.....	73
Referencias a Componentes	73
Parámetros de cascada	76
Ruteo.....	83
Nuevos códigos y limpieza	83
Page.....	88
App.razor.....	88
NavigationManager	90
Parámetros de ruta	92
NavLink	94
Lazy loading.....	95
Formularios	100
EditForm.....	100
OnSubmit	100
OnValidSubmit	100
OnInvalidSubmit.....	100
Validaciones	102
¿Salvar los cambios?	107
SweetAlert2.....	110
Filtro de películas.....	113
Formulario de actores.....	117
Insertar imagen	117
Componente de Markdown.....	120
Formulario de películas.....	124
Selección múltiple.....	127
Componente de selección múltiple	135
HttpClient.....	143

Entity Framework Core	145
DbContext	146
DBSet.....	147
Migraciones.....	154
CRUD	161
Crear géneros.....	161
Mensajes de Error	166
Crear actores.....	170
Guardado de imágenes	173
Azure Storage.....	174
Imágenes locales.....	176
Crear películas.....	180
Leyendo registros	191
Filtros	200
Visualizar película.....	211
Referencias cíclicas	212
Actualizaciones	219
Géneros.....	219
Actores	227
AutoMapper.....	229
Películas	235
Borrando registros	245
Géneros.....	247
Actores	251
Películas	257
Paginación.....	263
Filtros	274
Ejecución diferida.....	274
Algoritmo de Diferencias y @Key	281
Seguridad	285
Autenticación	286
AuthorizeView.....	291
Roles.....	297

Proteger componentes	303
Usuario autenticado.....	306
Identity.....	307
Construyendo JWT	320
Atributo Authorize	340
Componente de votación.....	348
Back-end del componente	356
Filtro de más votadas.....	371
Listado de usuarios	378
Roles.....	382
Renovando el JWT.....	405
Deslogueo automático	419
Despliegue.....	423
Blazor Server	426
Nueva app	430
Migrando usuarios	437
Migrando géneros.....	451
AsNoTracking	456
Migrando actores.....	456
Migrando películas.....	465
Migrando votación.....	485
Internacionalización.....	490
Globalización.....	490
Localización	490
Culture y Culture UI	491
Culture.....	491
Culture UI	492
QueryStrings.....	492
Cookies	492
HTTP Header	492
App multi-idioma	492
Cambio manual	504
Formato de fecha y números.....	510

Sin cambios en cultura.....	513
Traduciendo errores	516
Internacionalización en Blazor-Server	521
Selección manual	524
Progressive Web Apps	533
Diferencia entre PWA y una aplicación híbrida	534
Modo offline en desarrollo	538
Publish.....	538
Cacheando archivos	542
Modo offline	543
onInstall.....	550
onActivate	551
onFetch	551
Transformar app en PWA.....	555
Cacheando GETs.....	562
IndexedDB.....	577
Grabando géneros offline	587
Evento del Synchronizer	591
Borrando géneros offline	595
Notificaciones	599
Push API – Back-End.....	600
Push API I – Front-End.....	606

Blazor

Ventajas

- Ecosistema de .NET
- C# (LINQ y programación asíncrona)
- Compartir código entre front-end y back-end
- Trabajar con componentes

Tipos

WebAssembly (cliente)

Permite tener aplicaciones de .NET corriendo en el navegador. Esto implica descargar el Runtime en el navegador junto con las DLLs necesarias para correr nuestra aplicación. Hay 2 opciones:

1. Sólo crear el lado del cliente.
2. Alojar la app en ASP.NET Core.

Client-side

Sólo se utilizan archivos estáticos. Así, sólo tendremos un proyecto de front-end con código C# corriendo en el navegador que puede hacer peticiones HTTP hacia Web APIs realizadas en cualquier tecnología.

Client-side + ASP.NET Core

Se nos crea un proyecto de front-end y uno de back-end que va a servir como un web API para nuestra app. Es ideal cuando vamos a tener front y back-end que serán desplegados al mismo tiempo. Se puede compartir código entre front-end y back-end.

Ventaja: altamente escalable

Desventaja: Descargar runtime de .NET y DLLs podría implicar un download de 2MB.

Server-side

La app corre en el servidor. El cliente interactúa con ella a través de una conexión de SignalR. Con SignalR manejamos comunicación en tiempo real.

Esto hace entonces que el cliente no tenga que descargar el runtime de .NET, sino que simplemente interactúe con la aplicación de manera remota. Es claro que en este caso no tenemos la inherente descarga de los 2 MB para correr la aplicación, lo que hace que la aplicación servida del lado del servidor cargue más rápido.

Ventaja: dispositivos con menos recursos deberían poder correr la aplicación sin problemas. Pues el trabajo pesado queda delegado al servidor.

Desventajas: siempre necesitaremos un servidor disponible para correr la app. Esto significa que puede ser difícil servir a muchos usuarios desde un servidor de recursos ilimitados => latencia (si la conexión del cliente no fuera buena).

Híbrido

Nos permite tener aplicaciones que se ejecuten de manera nativa en ambientes desktop y móviles. Básicamente lo que se hace es tener una aplicación nativa, la cual utiliza un control de Web View para poder visualizar la aplicación de Blazor. Podemos utilizar Blazor híbrido con distintas tecnologías de .NET como Winforms, WPF y MAUI. Con las 2 primeras se pueden crear app desktop y utilizar componentes de Blazor en ellas. Con MAUI se pueden crear tanto app desktops como móviles.

Ventaja: poder compartir código entre aplicaciones web, desktop y móviles. Las app híbridas tienen acceso casi completo a las capacidades nativas del dispositivo.

Desventaja: Despliegues (Play Store).

Modelos de alojamiento

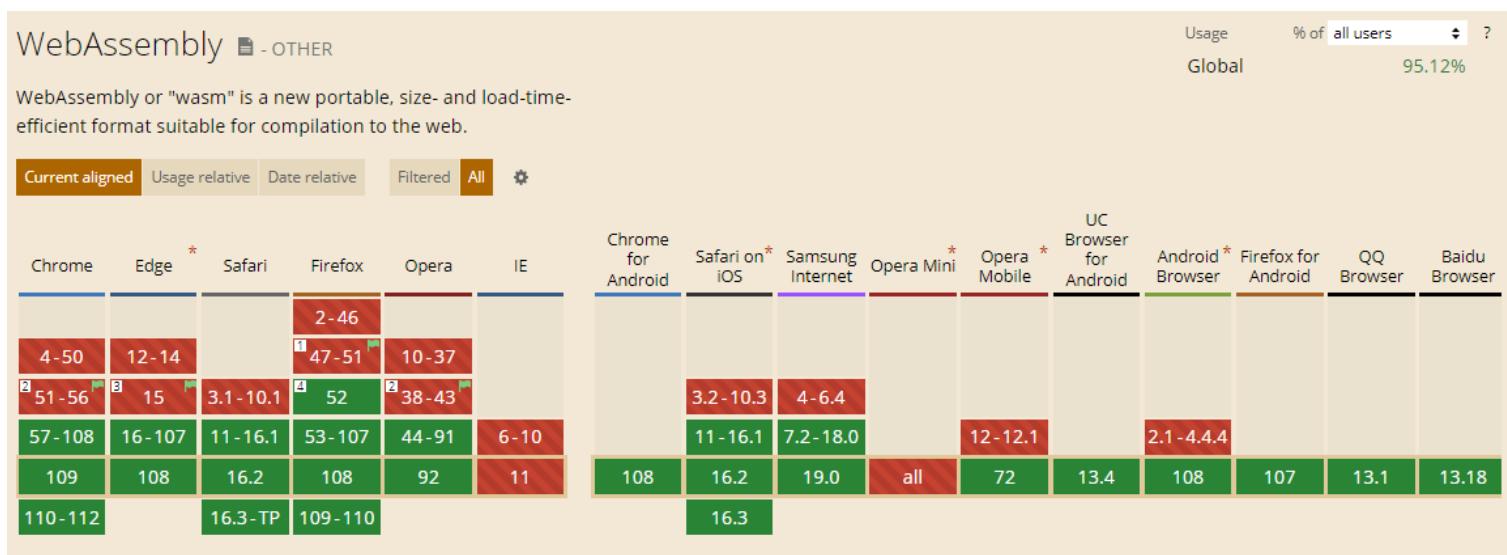
Característica	Blazor WebAssembly	Blazor del lado del servidor	Blazor híbrido
Utilizar código de C# y no JavaScript	✓	✓	✓
Descarga ligera de archivos	Se debe descargar el runtime de .NET + dependencias	✓	✓
Trabaja bien con dispositivos limitados	Todo el código debe ser cargado en el dispositivo	✓	Todo el código debe ser cargado en el dispositivo
Velocidad de ejecución	✓	Puede haber latencia	✓
Serverless	✓	Necesita un servidor	✓
Independiente de ASP.NET Core	✓	Requiere ASP.NET Core	✓
Independiente de WebAssembly	Requiere WebAssembly	✓	✓
Fácil escalabilidad	✓	Puede ser un reto	✓
Ser servida desde un CDN	✓	Necesita un servidor	N/A
Modo offline	✓	Necesita un servidor	✓
Acceso total a capacidades nativas del dispositivo	No, pues funciona a través de un navegador.	No, pues funciona a través de un navegador.	✓

Dónde se puede usar

WebAssembly

En navegadores que soporten WebAssembly.

<https://caniuse.com/?search=webassembly>



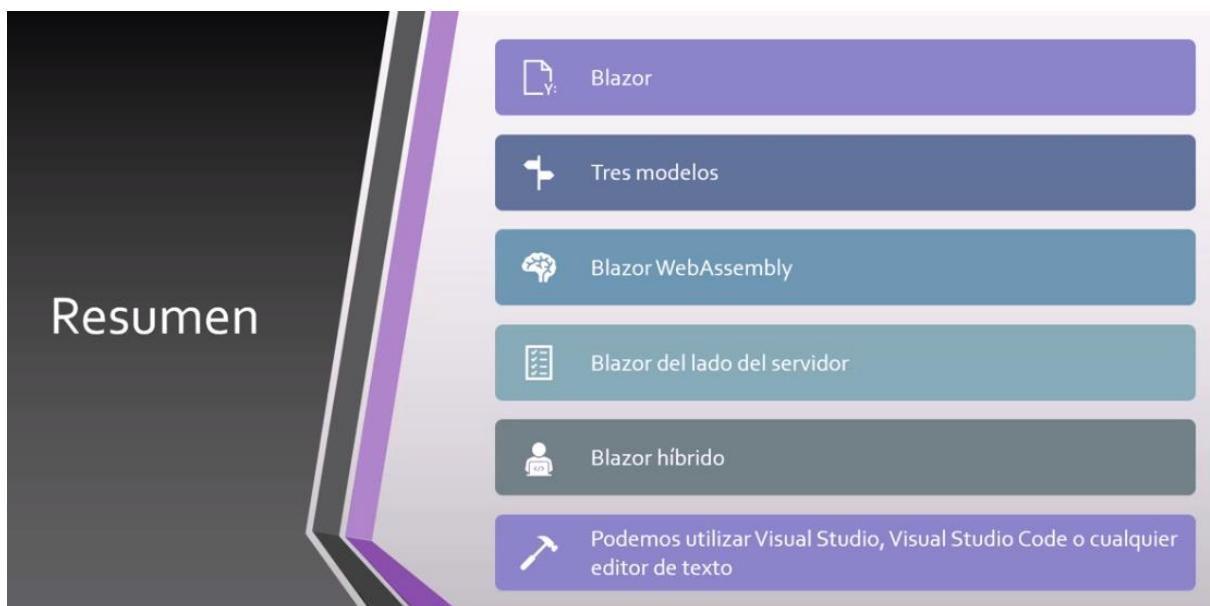
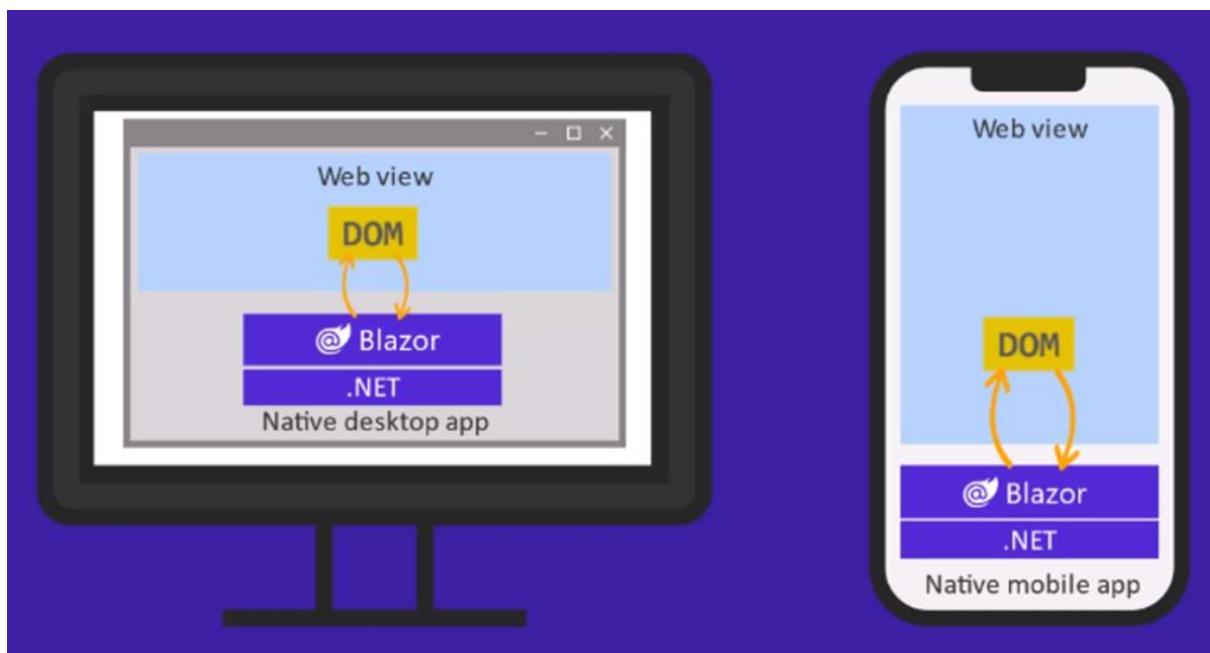
Al 16-Ene-2023, el 95.12% de los navegadores lo soportan:

Server-side

En el caso de server-side, todos los navegadores deberían servir porque el código se corre en el servidor. Sin embargo, la documentación de MS dice que en IE v11 se puede soportar pero sólo si se utilizar Polyfills. Las versiones anteriores ni siquiera soportan del lado del servidor.

Híbrido

Se puede utilizar en desktops/móviles.



Nueva app

Iniciamos una nueva. Buscamos Blazor y elegimos “Blazor WebAssembly App” (no la que dice empty). Tildamos “ASP.NET Core Hosted” que me permitirá tener un web api acompañando a mi app de Blazor (en esta accederemos a una BD).

Si quisiéramos hacerlo con Visual Studio Code, el comando para crear una nueva es:

```
dotnet new blazorwasm -n BlazorPeliculas -o BlazorPeliculas -ho
```

n: Name

o: Output

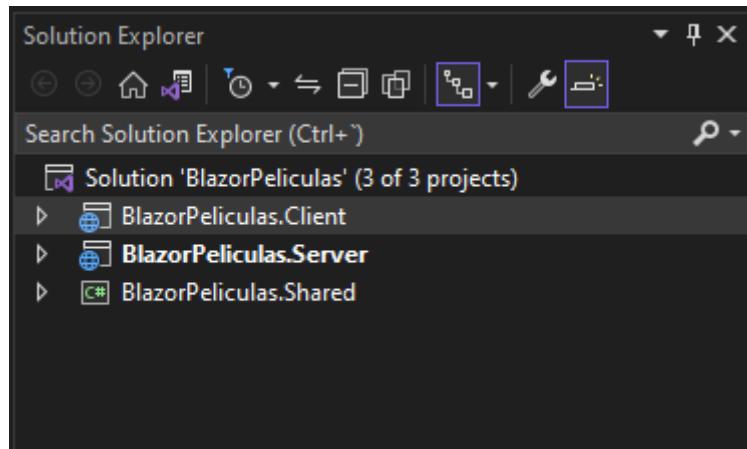
ho: hosted (webapi)

Una vez que se creó el proyecto con `code .` se levanta el Visual Studio Code desde esta carpeta.

Razor

Sintaxis que no permite combinar código HTML y código C#.

En nuestra app se puede ver que la solución consta de 3 proyectos: **BlazorPeliculas.Client**, **BlazorPeliculas.Server** y **BlazorPeliculas.Shared**.



En el archivo **BlazorPeliculas.Client/Pages/Index.razor** borramos todo y dejamos sólo el comando `@page`. Este sirve para indicar el URL con el cuál podremos acceder directamente al componente.

`@page`

Veamos el siguiente código.

```
@page "/"

<p>Hola, @nombre</p>

@code {
    string nombre = "Emiliano";
}
```

El comando `@page` me permite indicar como acceder al componente. De esta forma, lo transformarmos en un componente routeable.

Expresiones implícitas

@nombre se trata de una expresión implícita de Razor. También podríamos agregar .ToUpper() a la expresión implícita.

Una expresión implícita podría ser más compleja e incluir un llamado a una función. Por ejemplo:

```
<p>Hola, @Transformar(nombre)</p>

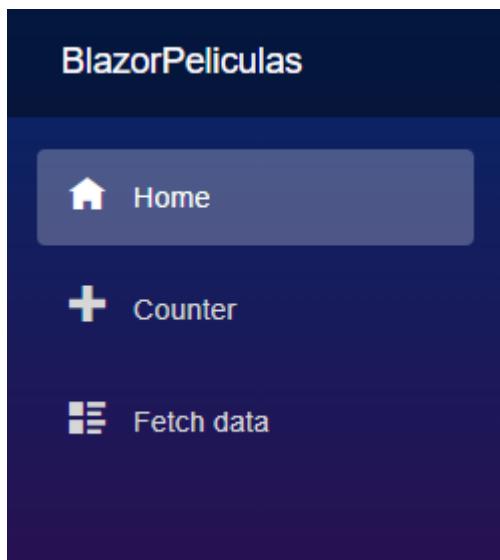
@code {
    string nombre = "Emiliano";

    string Transformar(string valor) => valor.ToUpper();
}
```

Expresiones explícitas

Las expresiones explícitas indican el lugar donde comienza y donde termina la expresión. Esto se hace con paréntesis. Por ej:

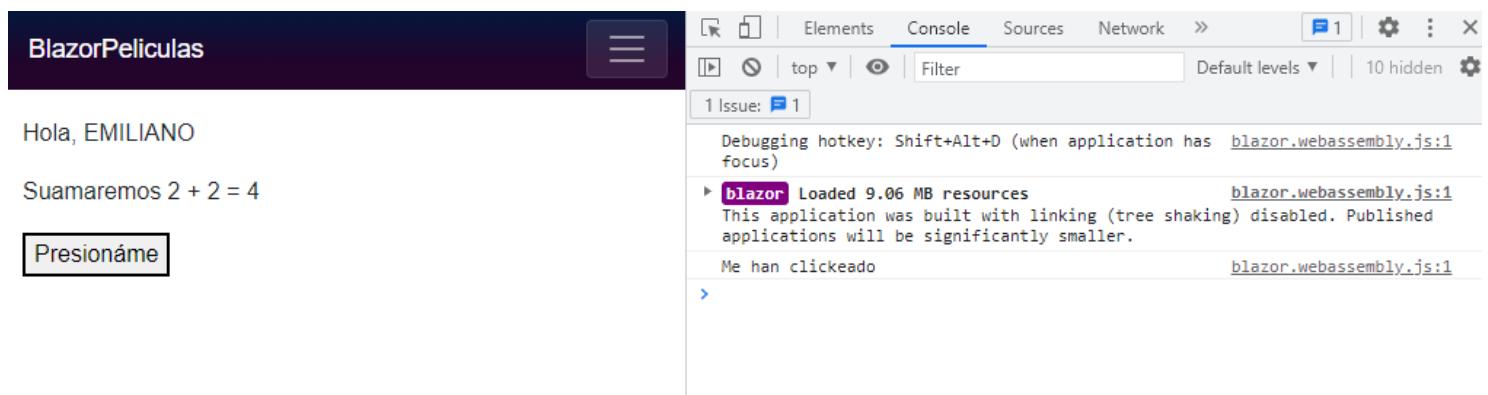
```
<p>Sumaremos 2 + 2 = @(2 + 2)</p>
```



Algo para lo cual utilizaremos bastante en este curso las expresiones explícitas es para utilizar expresiones lambda para definir el evento de un control de HTML. Por ejemplo:

```
<button @onclick=@(() => Console.WriteLine("Me han clickeado"))>Presionáme</button>
```

Con @onclick le indicamos que el código a ejecutar cuando se ejecuta el click es código C#. En este caso, el código a ejecutar en el click es una expresión explícita que consta de una función anónima que no recibe parámetros y cuyo cuerpo escribe una línea en la consola.



El curso no lo explica pero he probado que tanto onclick como @onclick produce el mismo efecto. Ya que el @() de la expresión explícita ya le indica que el código a correr es de C#:

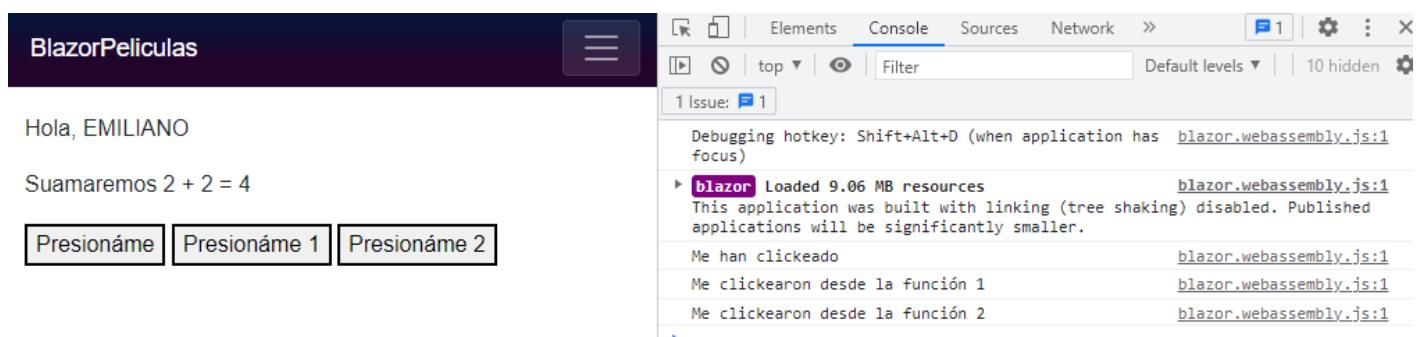
```
<button @onclick=@(() => Console.WriteLine("Me han clickeado"))>Presionáme</button>
<button @onclick=@ManejarClickBoton1>Presionáme 1</button>
<button onclick=@ManejarClickBoton2>Presionáme 2</button>

@code {
    string nombre = "Emiliano";

    string Transformar(string valor) => valor.ToUpper();

    void ManejarClickBoton1() {
        Console.WriteLine("Me clickearon desde la función 1");
    }

    void ManejarClickBoton2() {
        Console.WriteLine("Me clickearon desde la función 2");
    }
}
```



The screenshot shows a browser window titled "BlazorPelículas". In the top right corner, there's a developer tools panel with tabs for Elements, Console, Sources, Network, etc. The Console tab is selected. It displays a single warning message: "blazor Loaded 9.06 MB resources blazor.webassembly.js:1 This application was built with linking (tree shaking) disabled. Published applications will be significantly smaller." Below this, there are three log entries: "Me han clickeado blazor.webassembly.js:1", "Me clickearon desde la función 1 blazor.webassembly.js:1", and "Me clickearon desde la función 2 blazor.webassembly.js:1".

Si bien el código funciona, la sintaxis de `onclick=@ManejarClickBoton2` arroja un warning.

Clases

Algo muy útil es centralizar lógica en clases. Por ej., podemos crear centralizar las funciones de String en una clase de `StringUtilities`.

Index.razor

```
@page "/"
@using BlazorPeliculas.Client.Utilities

<p>Hola, @StringUtilities.Transformar(nombre)</p>

@code {
    string nombre = "Emiliano";
}
```

StringUtilities.cs

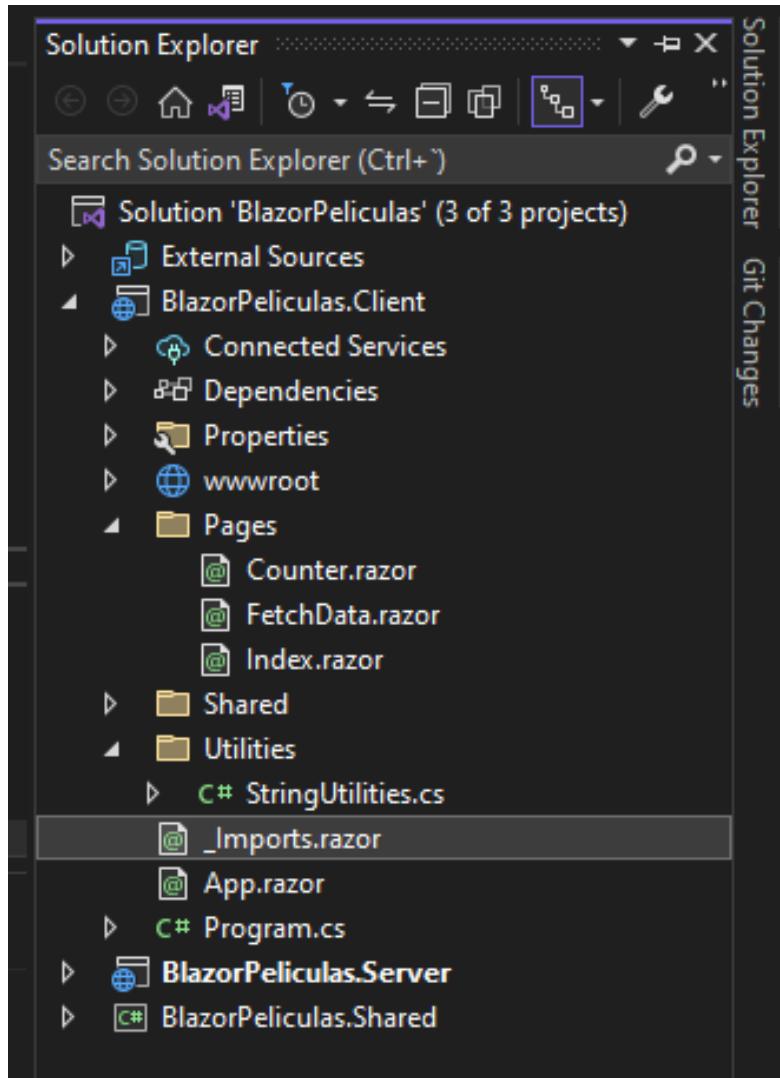
```
namespace BlazorPeliculas.Client.Utilities {
    public class StringUtilities {
        public static string Transformar(string valor) => valor.ToUpper();

    }
}
```

Hemos declarado la función pública para que sea accesible desde afuera y estática para que no sea necesario tener una instancia de la clase `StringUtilities` para poder utilizarla.

_Imports.razor

Puede suceder que estas funciones vayan a ser utilizadas desde muchísimos componentes. Para evitar tener que agregar el using en cada uno de ellos, se lo puede agregar directamente en _Imports.razor.



Index.razor

```
@page "/"

<p>Hola, @StringUtilities.Transformar(nombre)</p>

@code {
    string nombre = "Emiliano";
}
```

_Imports.razor

```
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
```

```
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorPeliculas.Client
@using BlazorPeliculas.Client.Shared
@using BlazorPeliculas.Client.Utilities
```

Proyectos

Client es el proyecto que se descargará en el dispositivo. Es lo que el usuario verá y con lo que interactuará. Es la capa de presentación.

Server es el proyecto que queremos tener en el servidor, representa mi webapi. Todo lo que sea sensible, tiene que estar aquí. Por ej: la lógica para acceder a la base de datos. Esto no se descargará en el navegador del usuario.

Shared es para poder compartir información entre los 2 proyectos anteriores.

Al declarar propiedades se puede agregar **= null!**; al final para indicar que tal propiedad puede ser nula (al salir de la construcción del objeto). Otra forma es poner **?** detrás del tipo de dato. Eso significa que está bien que el dato sea nulo. La diferencia entre ambos es que el primero le perdona el nulo y el segundo indica que el valor puede ser nulo.

foreach

Movie.cs

```
namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public string Title { get; set; } = null!;
        public DateTime ReleaseDate { get; set; }

    }
}
```

Index.razor

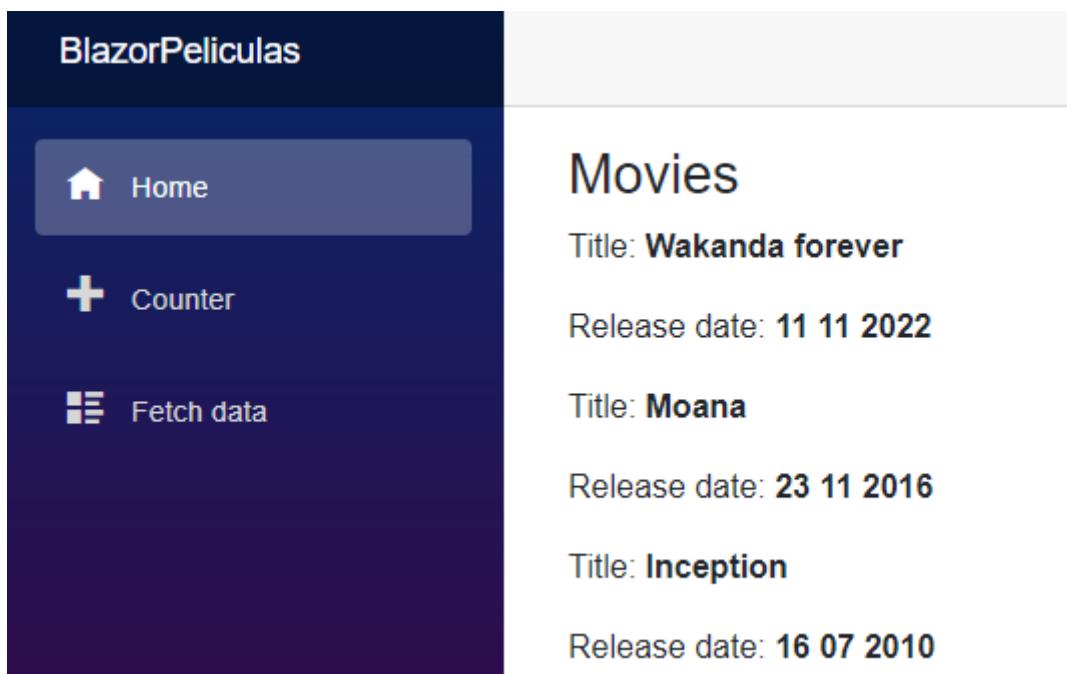
```
@page "/"

<div>
    <h3>Movies</h3>

    @foreach(var movie in GetMovies()) {
        <div>
            <p>Title: <b>@movie.Title</b></p>
            <p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
        </div>
    }
}
```

```
</div>

@code {
    List<Movie> GetMovies() {
        return new List<Movie>() {
            new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
            new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
            new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
        };
    }
}
```



For

Index.razor

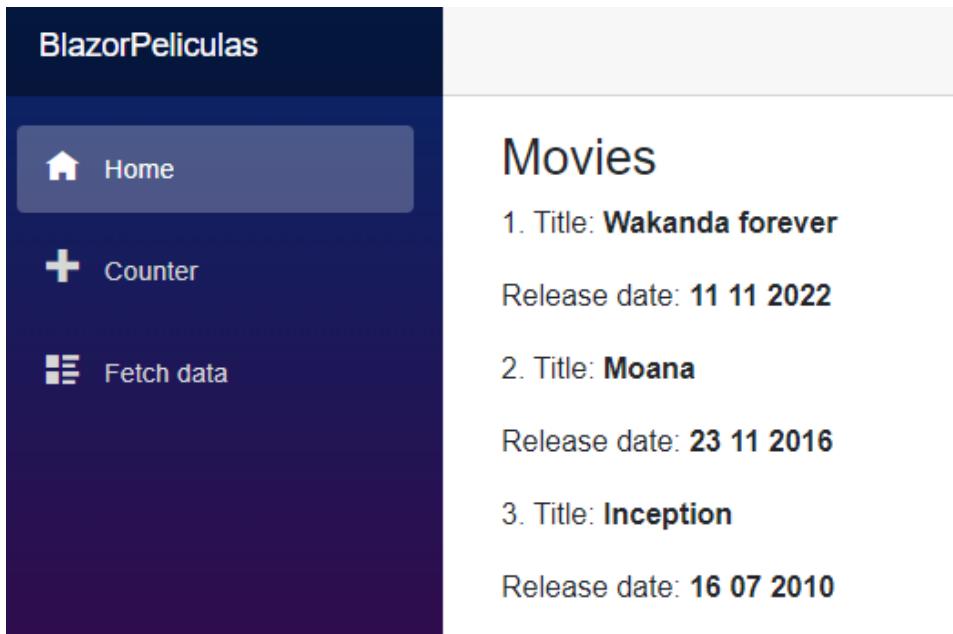
```
@page "/"

<div>
    <h3>Movies</h3>
    @for (int i = 0; i < Movies.Count; i++) {
        <div>
            <p>@(i+1). Title: <b>@Movies[i].Title</b></p>
            <p>Release date: <b>@Movies[i].ReleaseDate.ToString("dd MM yyyy")</b></p>
        </div>
    }
</div>

@code {
    List<Movie> Movies { get { return GetMovies(); }}}
```

```
List<Movie> GetMovies() {
    return new List<Movie>() {
        new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
        new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
        new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
    };
}
```

Básicamente, la principal diferencia (o ventaja) es la posibilidad de acceder al índice (i).



The screenshot shows a Blazor application interface. On the left is a sidebar with three items: 'Home' (selected), 'Counter', and 'Fetch data'. The main content area has a title 'Movies'. Below it, there is a list of three movies:

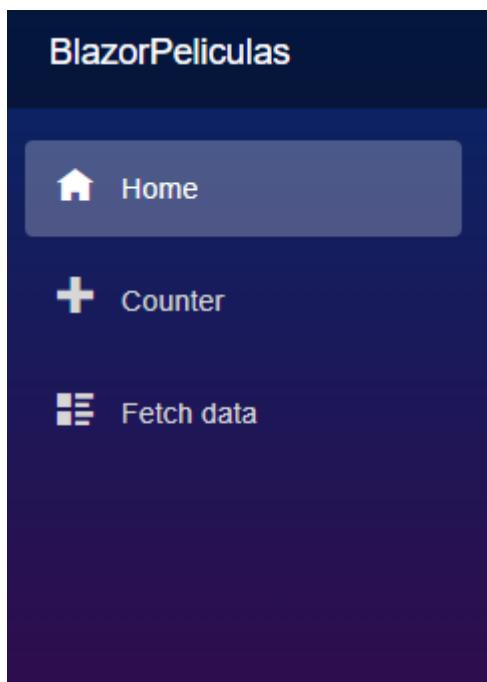
- 1. Title: **Wakanda forever**
Release date: **11 11 2022**
- 2. Title: **Moana**
Release date: **23 11 2016**
- 3. Title: **Inception**
Release date: **16 07 2010**

Condicionales

Los condicionales nos permite ejecutar código si se cumple la condición. En nuestro caso, agregaremos (**New!**) si se trata de una película cuyo lanzamiento sea menor a los 3 meses. Al 17/01/2023, **Wakanda forever** es nueva porque se estrenó hace menos de 3 meses.

Index.razor

```
@foreach(var movie in GetMovies()) {
    <div>
        <p>Title: <b>@movie.Title</b>
        @if(DateTime.Today.Subtract(movie.ReleaseDate).Days <= 31*3) {
            <span style="color:red"> (New!)</span>
        }
        </p>
        <p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
    </div>
}
```



Movies

Title: **Wakanda forever (New!)**

Release date: **11 11 2022**

Title: **Moana**

Release date: **23 11 2016**

Title: **Inception**

Release date: **16 07 2010**

Espera

Simularemos una espera en la carga de las películas (que sería lo más lógico si la lista tuviera que traerse desde un servidor remoto). Para ello, simularemos una demora de 3 segundos y que la variable Movies esté en null al comienzo. Para conseguir esto, eliminamos el **get** que hacia el return de GetMovies y agregamos el **set**. Para evitar que el IntelliSense marque el error de que la variable es nula, agregamos el **?** detrás del tipo de dato.

Index.razor

```
@page "/"
```

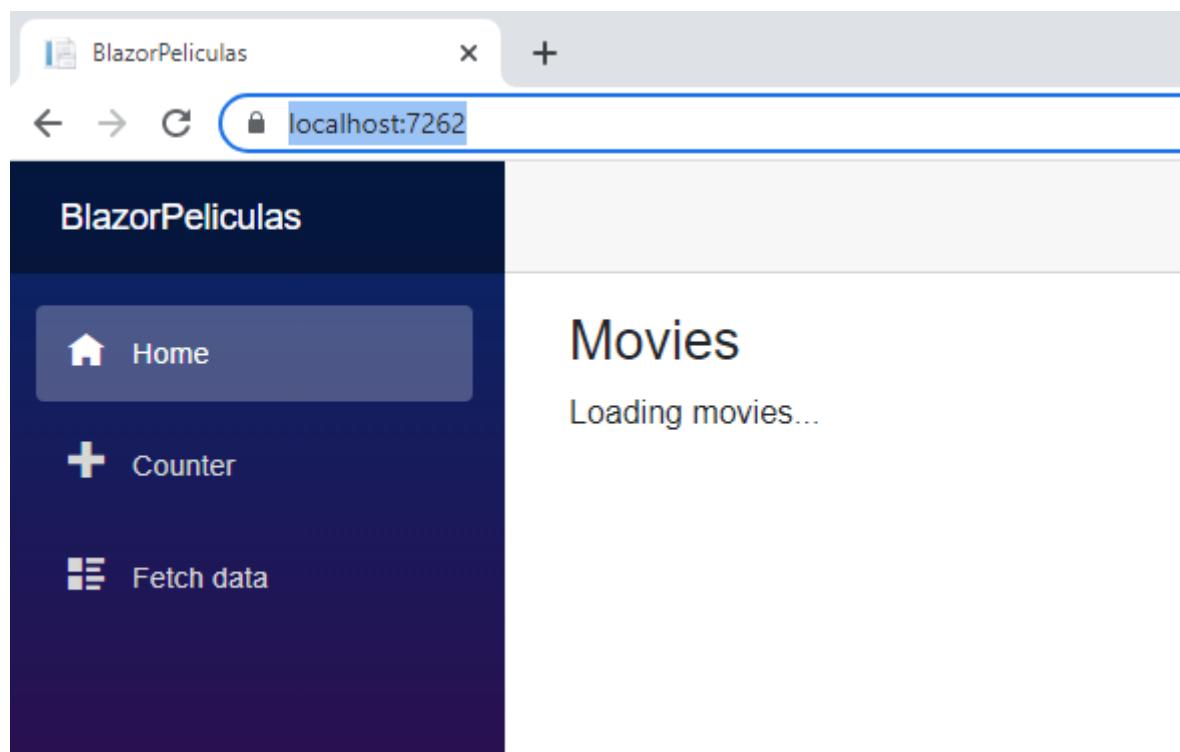
```
<div>
    <h3>Movies</h3>
    @if(Movies is null) {
        <p>Loading movies...</p>
    }
    else
        foreach(var movie in Movies) {
            <div>
                <p>Title: <b>@movie.Title</b>
                @if(DateTime.Today.Subtract(movie.ReleaseDate).Days <= 31*3) {
                    <span style="color:red"> (New!)</span>
                }
                </p>
                <p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
            </div>
        }
    }
```

```
</div>

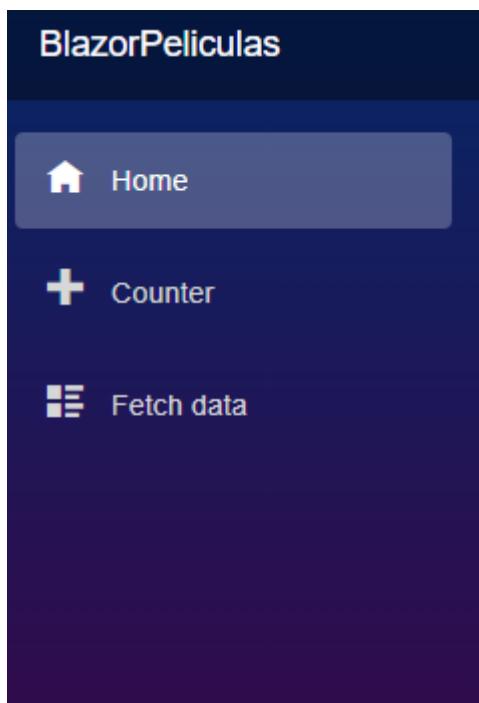
@code {
    List<Movie>? Movies { get; set; }

    protected override async Task OnInitializedAsync() {
        await Task.Delay(3000);
        Movies = GetMovies();
    }

    List<Movie> GetMovies() {
        return new List<Movie>() {
            new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
            new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
            new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
        };
    }
}
```



Y 3 segundos después...



Incluso, podemos tener un else if para mostrar que no tenemos películas para mostrar:

```
Index.razor
@page "/"

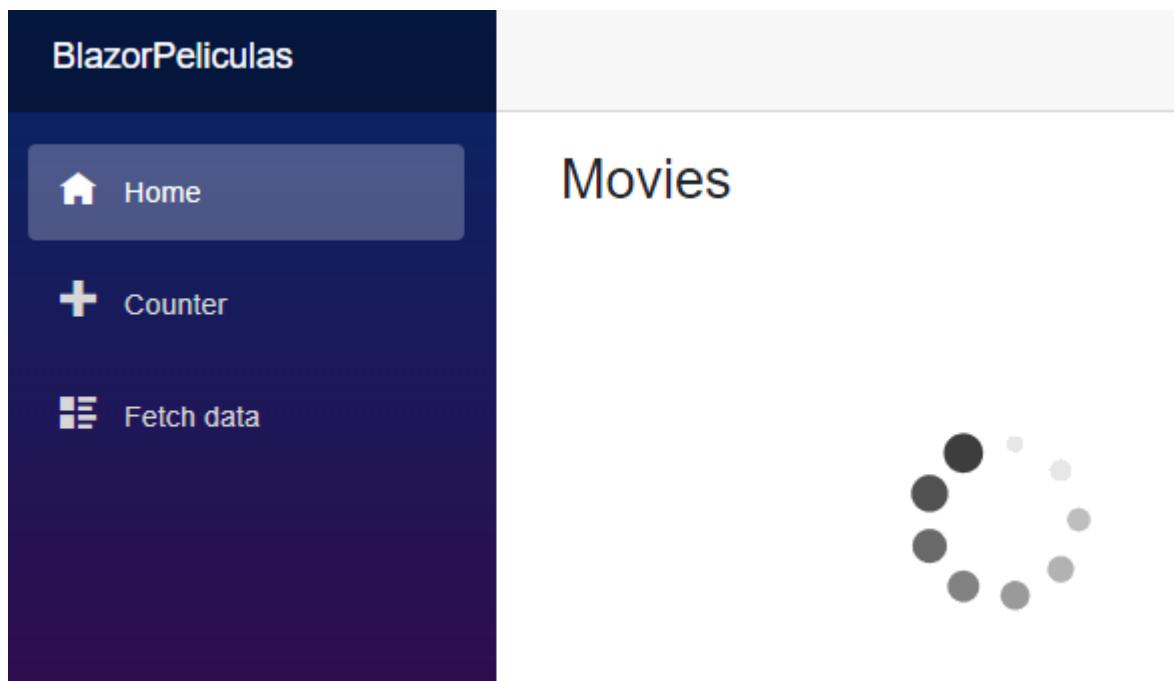
<div>
    <h3>Movies</h3>
    @if(Movies is null) {
        
    }
    else if(Movies.Count == 0) {
        <p>No movies to show</p>
    }
    else
        foreach(var movie in Movies) {
            <div>
                <p>Title: <b>@movie.Title</b></p>
                @if(DateTime.Today.Subtract(movie.ReleaseDate).Days <= 31*3) {
                    <span style="color:red"> (New!)</span>
                }
                </p>
                <p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
            </div>
        }
    }
}
```

```
</div>

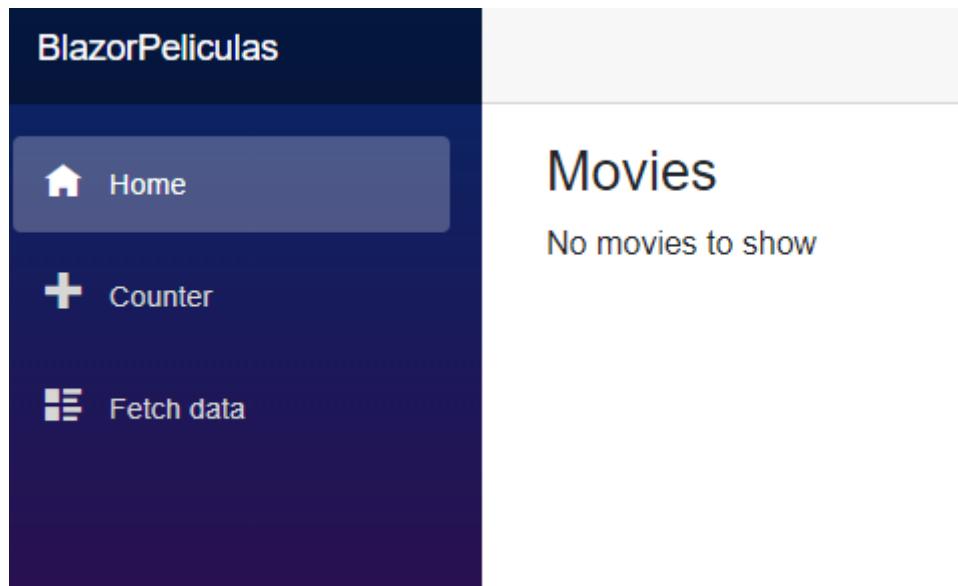
@code {
    List<Movie>? Movies { get; set; }

    protected override async Task OnInitializedAsync() {
        await Task.Delay(3000);
        Movies = new List<Movie>();
    }

    List<Movie> GetMovies() {
        return new List<Movie>() {
            new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
            new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
            new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
        };
    }
}
```



Y 3 segundos después



Comentarios

Para comentar grupo de líneas se usa **Ctrl+K+C** que agrega n **@*** al comienzo y un ***@** al final.

MarkupString

Se puede imprimir código HTML que viniera desde la BD (por ejemplo). Por una medida de seguridad, para hacer esto es necesario castear el texto como MarkupString para evitar que código malicioso se ejecute automáticamente. Por lo tanto, tenemos que estar seguros de que el código al que castearemos como Markup está libre de código malicioso antes de permitir que corra en el navegador como tal.

Index.razor

```
@page "/"

<div>
    <h3>Movies</h3>
    @if(Movies is null) {
        
    }
    else if(Movies.Count == 0) {
        <p>No movies to show</p>
    }
    else
        foreach(var movie in Movies) {
            <div>
```

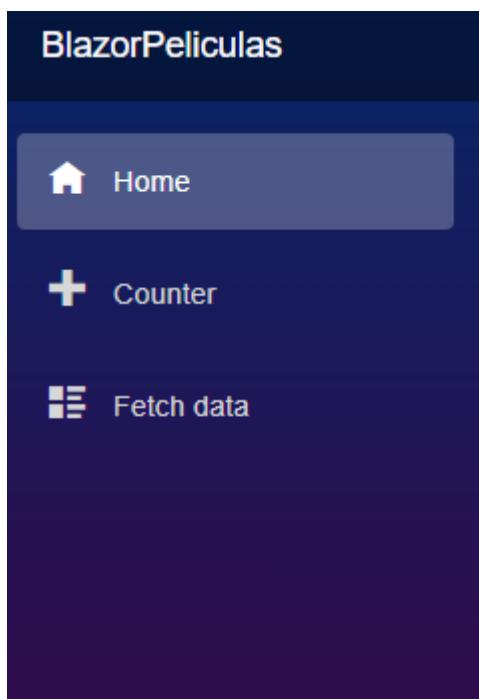
```
<p>Title: @((MarkupString)movie.Title)
@if(DateTime.Today.Subtract(movie.ReleaseDate).Days <= 31*3) {
    <span style="color:red"> (New!)</span>
}
</p>
<p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
</div>
}

</div>

@code {
List<Movie>? Movies { get; set; }

protected override async Task OnInitializedAsync() {
    await Task.Delay(3000);
    Movies = GetMovies();
}

List<Movie> GetMovies() {
    return new List<Movie>() {
        new Movie {Title = "<b>Wakanda forever</b>", ReleaseDate = new DateTime(2022, 11, 11) },
        new Movie {Title = "<i>Moana</i>", ReleaseDate = new DateTime(2016, 11, 23) },
        new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16) }
    };
}
}
```



Movies

Title: Wakanda forever (New!)

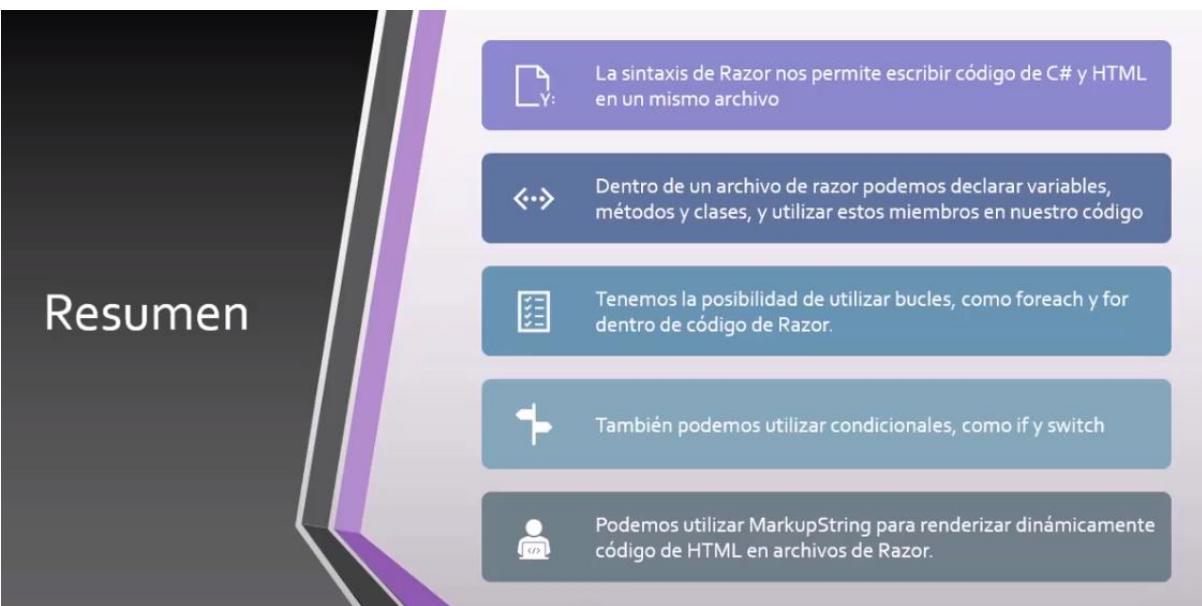
Release date: 11 11 2022

Title: Moana

Release date: 23 11 2016

Title: Inception

Release date: 16 07 2010



Resumen

-  La sintaxis de Razor nos permite escribir código de C# y HTML en un mismo archivo
-  Dentro de un archivo de razor podemos declarar variables, métodos y clases, y utilizar estos miembros en nuestro código
-  Tenemos la posibilidad de utilizar bucles, como foreach y for dentro de código de Razor.
-  También podemos utilizar condicionales, como if y switch
-  Podemos utilizar MarkupString para renderizar dinámicamente código de HTML en archivos de Razor.

Componentes

Son el corazón de una app en Blazor. Casi todo es un componente: interfaz de usuario re-utilizable que puede tener lógica. Aunque no lo parezca, es una clase.

Routeables

Los componentes ruteables tienen la directiva `@page` que le indica a Blazor la ruta con la que puede accederse directamente. Por lo general, estos se encuentran en la carpeta **Pages**.

Non-Routeables

Los componentes no ruteables NO tienen la directiva `@page`. Por lo general, estos se encuentran en la carpeta **Shared**.

Nuevo componente

Agregaremos un nuevo componente en **Shared**. Desde la carpeta, click-derecho **add -> Razor Component....** Lo nombraremos **MoviesList.razor**. Es importante que todo componente razor comience con mayúscula.

Index.razor

```
@page "/"

<div>
    <h3>Movies</h3>
    <MoviesList />
</div>

@code {
```

MoviesList.razor

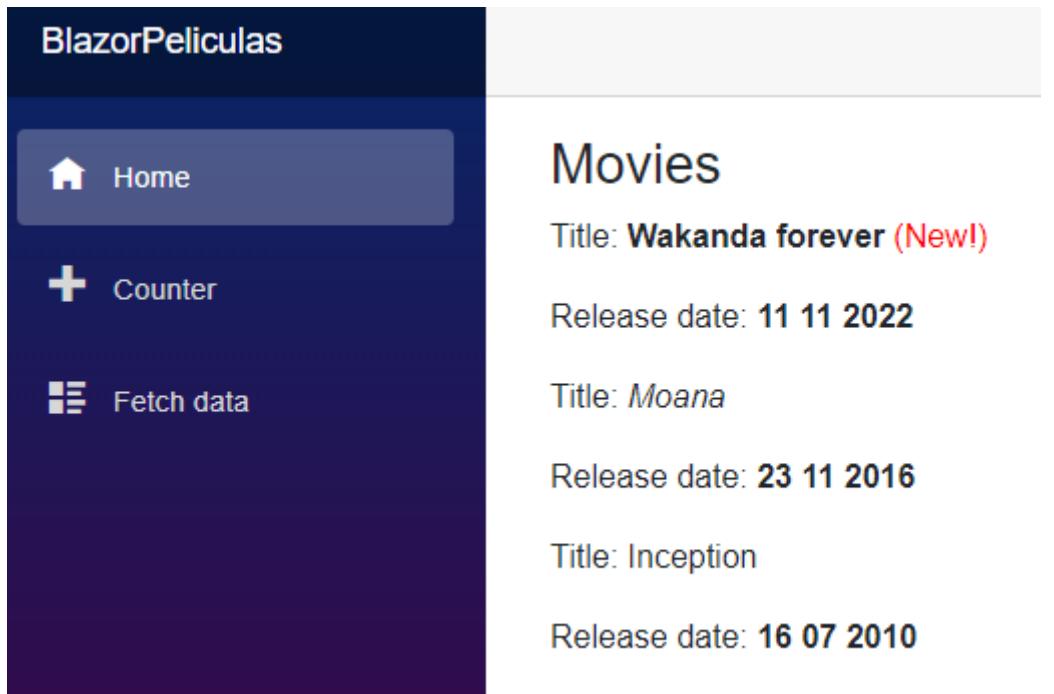
```
@if(Movies is null) {
    
}
else if(Movies.Count == 0) {
    <p>No movies to show</p>
}
else
    foreach(var movie in Movies) {
        <div>
            <p>Title: @movie.Title
                @if(DateTime.Today.Subtract(movie.ReleaseDate).Days <= 31*3) {
                    <span style="color:red"> (New!)</span>
```

```
        }
    </p>
    <p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
</div>
}

@code {
public List<Movie>? Movies { get; set; }

List<Movie> GetMovies() {
    return new List<Movie>() {
        new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
        new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
        new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
    };
}

protected override async Task OnInitializedAsync() {
    await Task.Delay(3000);
    Movies = GetMovies();
}
}
```



```

<div>
    <h3>Movies</h3>
    @if(Movies is null) {
        
    }
    else if(Movies.Count == 0) {
        <p>No movies to show</p>
    }
    else
        foreach(var movie in Movies) {
            <div>
                <p>Title: @((MarkupString)movie.Title)
                    @if(DateTime.Today.Subtract(movie.ReleaseDate).Days <= 31*3) {
                        <span style="color:red"> (New!)</span>
                    }
                    </p>
                    <p>Release date: <b>@movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
            </div>
        }
    </div>
}

@code {
    List<Movie>? Movies { get; set; }

    protected override async Task OnInitializedAsync() {
        await Task.Delay(3000);
        Movies = GetMovies();
    }

    List<Movie> GetMovies() {
        return new List<Movie>() {
            new Movie {Title = "<b>Wakanda forever</b>", ReleaseDate = new DateTime(2022, 11, 11)},
            new Movie {Title = "<i>Moana</i>", ReleaseDate = new DateTime(2016, 11, 23)},
            new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
        };
    }
}

```

Parámetros

Los parámetros son variables que nos permite que nuestro componente reciba datos desde otro componente. Para ello, se usa la directiva [Parameter] y su variable debe ser declarada con scope público. Por ejemplo:

```

[Parameter]
public List<Movie>? Movies { get; set; }

```

Obligatorios

Si necesitamos definir un parámetro como obligatorio, usamos la directiva [EditorRequired] Por ejemplo:

```
[Parameter]
[EditorRequired]
public Movie Movie { get; set; } = null!;
```

Esto no impide que la app compile pero sí que muestre una advertencia al compilar. A su vez, si no se agregara el parámetro, se marcaría el "error" en el IntelliSense pero permite su compilación.

Arbitrarios

Cuando tenemos un gran cantidad de parámetros que tendríamos que pasar a un componente (por ejemplo, un INPUT de HTML podría recibir class, disabled, type, checked (si fuera de tipo="checkbox"), value, etc.). Por ej:

```
Index.razor
@page "/"

<div>
    <h3>Movies</h3>
    <MoviesList />
    <EjTextbox placeholder="Nombre Película" value="Back to the future"></EjTextbox>
</div>

@code {
```

```
EjTextbox.razor
<input type="text" @attributes="AdditionalParameters"/>

@code {
    [Parameter(CaptureUnmatchedValues = true)]
    public IDictionary<string, object> AdditionalParameters { get; set; } = null!;
}
```

Otras 2 formas de pasar parámetros arbitrarios:

```
Index.razor
@page "/"

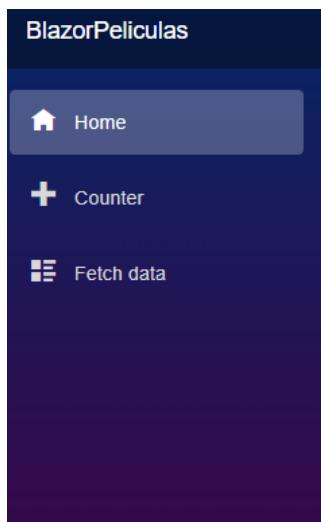
<div>
    <h3>Movies</h3>
    <MoviesList />
    <EjTextbox placeholder="Nombre Película" value="Back to the future"></EjTextbox>
```

```

<EjTextbox @attributes="@(new Dictionary<string, object>()
{
    { "placeholder", "Movie's title"}})"></EjTextbox>

<EjTextbox @attributes="ParametersSample"></EjTextbox>
</div>

@code {
    private Dictionary<string, object> ParametersSample = new Dictionary<string, object>()
    {
        { "placeholder", "Movie's title 2"}};
}
    
```



Movies

Title: *Wakanda forever* (New!)

Release date: 11 11 2022

Title: *Moana*

Release date: 23 11 2016

Title: *Inception*

Release date: 16 07 2010

[Back to the future](#)

Parámetros por defecto

Se pueden definir valores por defecto a los parámetros. En Index.razor crearemos dos EjTextBoxes. Al primero no le pondremos valor al parámetro placeholder y al segundo, sí lo haremos:

Index.razor

```

@page "/"

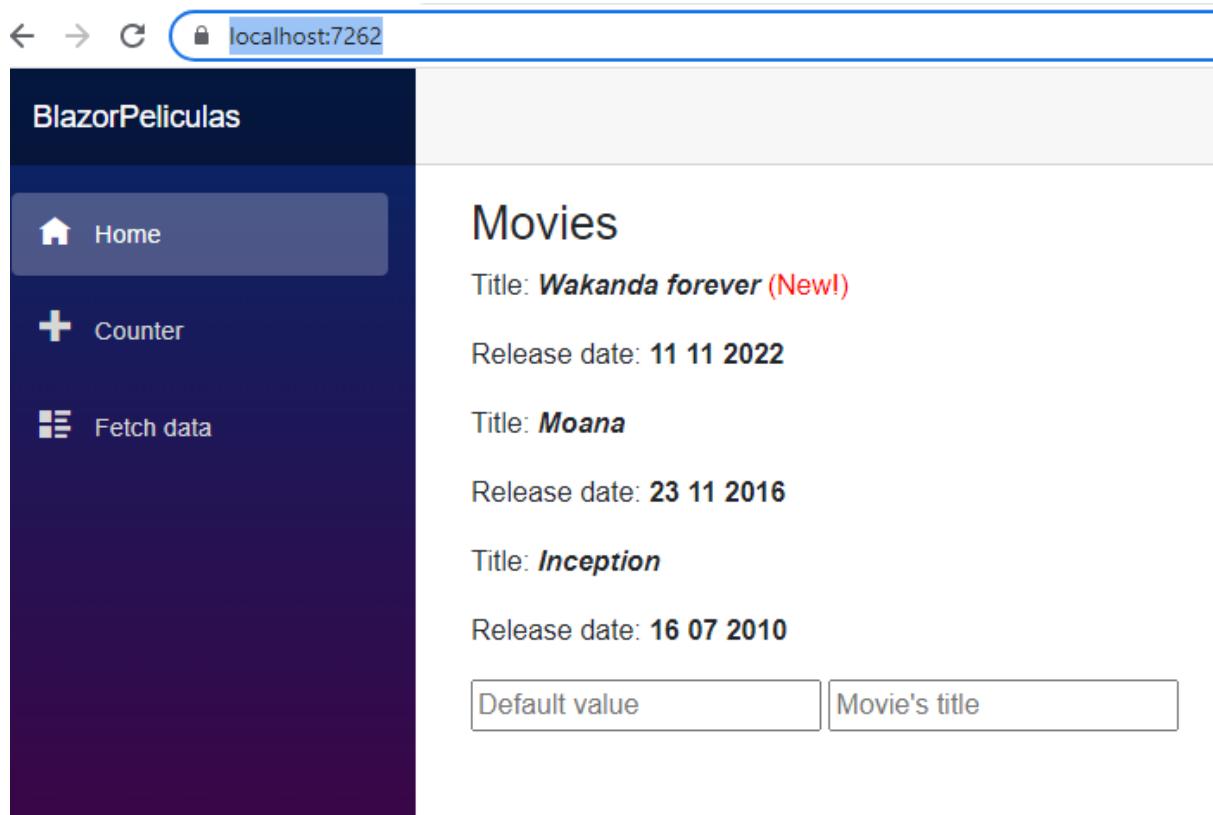
<div>
    <h3>Movies</h3>
    <MoviesList />
    <EjTextbox /></EjTextbox>
    <EjTextbox placeholder="Movie's title"/></EjTextbox></div>

@code {
}
    
```

EjTextbox.razor

```
<input type="text" placeholder="Default value" @attributes="AdditionalParameters"/>

@code {
    [Parameter(CaptureUnmatchedValues = true)]
    public IDictionary<string, object> AdditionalParameters { get; set; } = null!;
}
```



Es muy importante el orden ya que Blazor asigna la precedencia de derecha a izquierda. Si ponemos primero el `@attributes` y después el valor por defecto Blazor asignará el valor por defecto **SIEMPRE** (mayor precedencia).

Eventos

Son acciones que se ejecutan en respuesta a una interacción. Por ejemplo, el evento `click` se dispara cuando hacemos clic sobre un elemento con Blazor, o podemos ejecutar métodos en respuesta a eventos.

MovieItem.razor

```
<div>
    <p>Title: <b><i>@Movie.Title</i></b>
    @if(DateTime.Today.Subtract(Movie.ReleaseDate).Days <= 31*3) {
        <span style="color:red"> (New!)</span>
    }
```

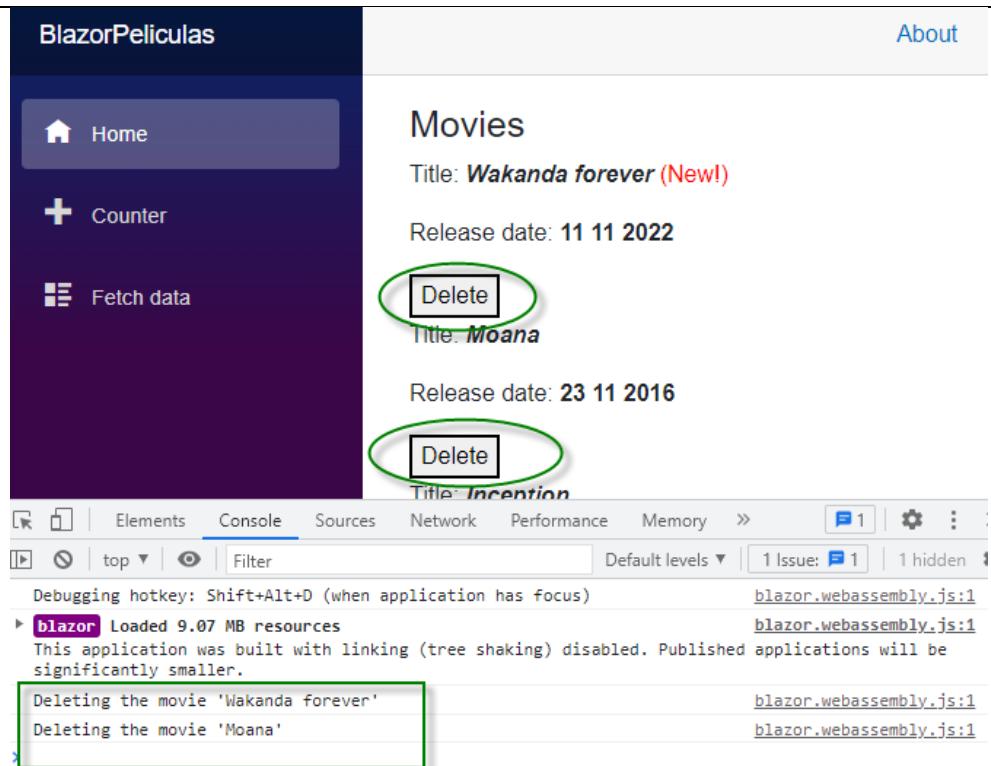
```

</p>
<p>Release date: <b>@Movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
<div>
    <button @onclick="DeleteMovie">Delete</button>
</div>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    void DeleteMovie() {
        Console.WriteLine($"Deleting the movie '{Movie.Title}'");
    }
}

```



Para mostrar/ocultar botones

MovieItem.razor

```

<div>
    <p>Title: <b><i>@Movie.Title</i></b>
        @if(DateTime.Today.Subtract(Movie.ReleaseDate).Days <= 31*3) {
            <span style="color:red"> (New!)</span>
        }
    </p>
    <p>Release date: <b>@Movie.ReleaseDate.ToString("dd MM yyyy")</b></p>

```

```

@if(ShowButtons) {
    <div>
        <button @onclick="DeleteMovie">Delete</button>
    </div>
}
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;

    void DeleteMovie() {
        Console.WriteLine($"Deleting the movie '{Movie.Title}'");
    }
}

```

MoviesList.razor

```

@if(Movies is null) {
    
}
else if(Movies.Count == 0) {
    <p>No movies to show</p>
}
else {
    <input type="checkbox" @onchange="@(() => ShowButtons = !ShowButtons)" />
    <span>Show buttons</span>
    @foreach(var movie in Movies) {
        <MovieItem Movie="movie" ShowButtons="ShowButtons"/>
    }
}

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }

    bool ShowButtons = false;
}

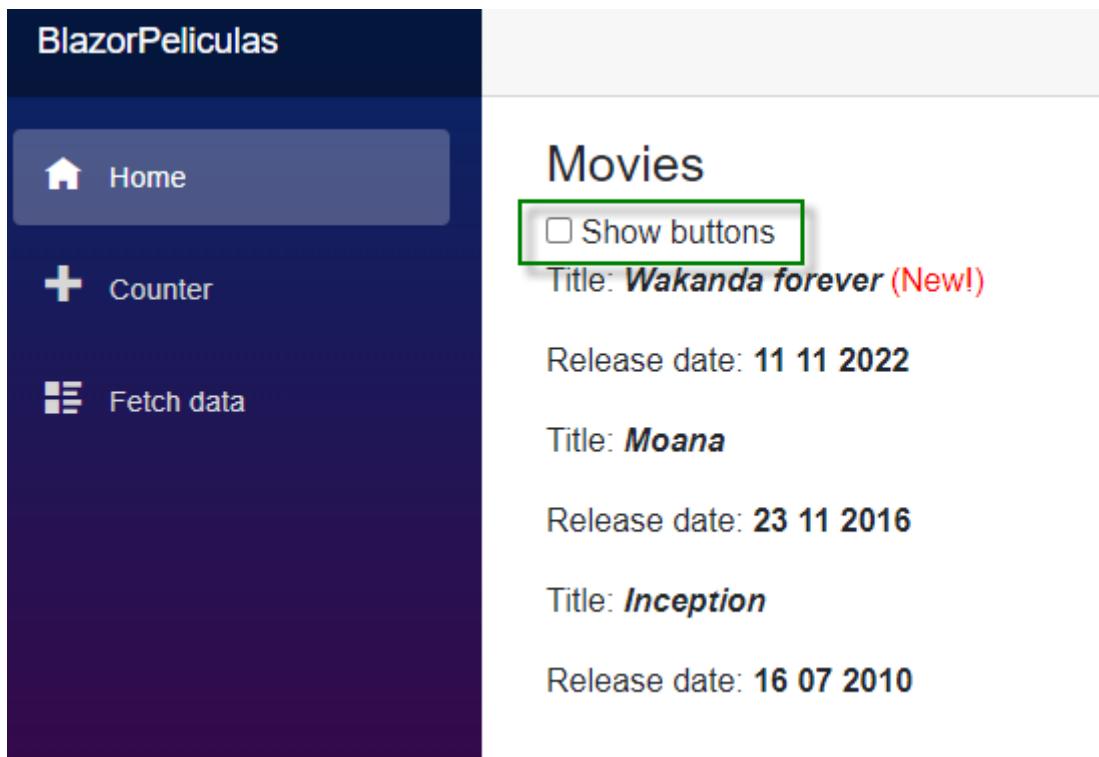
```

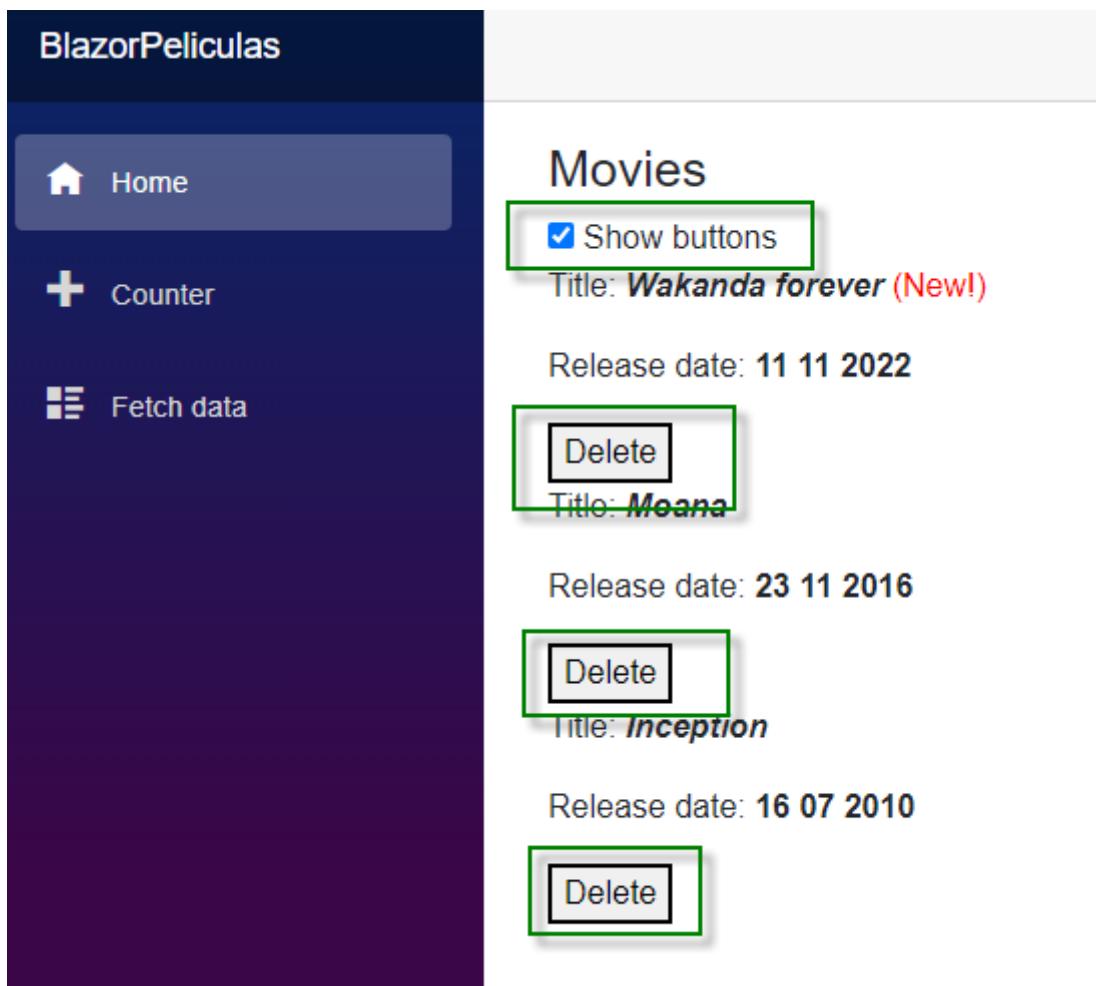
Index.razor

```
@page "/"
```

```
<div>
    <h3>Movies</h3>
    <div>
        <MoviesList Movies="GetMovies()" />
    </div>
</div>

@code {
    List<Movie> GetMovies() {
        return new List<Movie>() {
            new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
            new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
            new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
        };
    }
}
```





The screenshot shows a Blazor application titled "BlazorPelículas". On the left, there's a navigation menu with three items: "Home", "Counter", and "Fetch data". The main content area is titled "Movies" and contains a list of movies. Each movie entry includes its title, release date, and a "Delete" button. The first movie listed is "Wakanda forever (New!)" with a release date of "11 11 2022". The second movie is "Moana" with a release date of "23 11 2016". The third movie is "Inception" with a release date of "16 07 2010". The "Delete" button for each movie is highlighted with a green border.

Title	Release Date	Action
<i>Wakanda forever (New!)</i>	11 11 2022	Delete
<i>Moana</i>	23 11 2016	Delete
<i>Inception</i>	16 07 2010	Delete

Una mejor manera de conseguir esto es con Data Binding.

Data Binding

El Data Binding nos permite establecer un canal de sincronización entre una variable y un elemento HTML o un componente.

MoviesList.razor

```
@if(Movies is null) {
    
}
else if(Movies.Count == 0) {
    <p>No movies to show</p>
}
else {
    <input type="checkbox" @bind="ShowButtons" />
    <span>Show buttons</span>
    @foreach(var movie in Movies) {
        <MovieItem Movie="movie" ShowButtons="ShowButtons"/>
    }
}

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }

    bool ShowButtons = false;
}
```

Con eso sólo, el funcionamiento es igual al anterior.

Bind Event

También se puede definir el evento con el cuál se disparará la sincronización entre el elemento y la variable.

BindEvent.razor

```
<h3>BindEvent</h3>

<p>
    <input @bind="value" />
</p>
The value is: @value

@code {
    private string? value { get; set; }
}
```

BindEvent

```
Emilian
```

The value is:

BindEvent

```
Emiliano F.
```

The value is: Emiliano F.

El texto no cambia hasta que no salgo del textbox porque el bind, por defecto, se dispara con el onchange.

BindEvent.razor

```
<h3>BindEvent</h3>

<p>
  <input @bind="value" @bind:event="oninput" />
</p>
The value is: @value

@code {
    private string? value { get; set; }
}
```

Con eso sólo ya alcanza para que la variable value cambie su valor con cada tecla presionada.

Bind Get y Bind Set

Se puede tener más control a la hora de hacer el Binding. **@bind** es un sincronizador de doble sentido: tengo 2 operaciones (get y set). Esto es de .NET 7. Al momento de hacer estas pruebas, estoy con .NET 6 y no funciona como en el video:

BindGetSetDemo.razor

```
<h3>BindGetSetDemo</h3>
<input type="text" @bind:get="text" @bind:set="SetAsync"/>
<p>
  Text: @text
</p>

@code {
    private string text = string.Empty;

    private Task SetAsync(string value) {
        text = value.ToUpper();
        return Task.CompletedTask;
    }
}
```

Bind After

Es otro modificador que ofrece Blazor. Permite ejecutar una función después de realizar el binding. Es muy útil en el caso de buscadores para conseguir realizar la búsqueda después de ingresar el texto a buscar. Esto es de .NET 7. Al momento de hacer estas pruebas, estoy con .NET 6 y no funciona como en el video:

BindAfterDemo.razor

```
<h3>BindAfterDemo</h3>

<input type="text" @bind="searchText" @bind:after="Search"/>
<p>
    Found people:
</p>

<ul>
    @foreach(var people in foundPeople) {
        <li>@people</li>
    }
</ul>

@code {
    private string searchText = string.Empty;
    private List<string> foundPeople = new List<string>();

    private async Task Search() {
        var people = new List<string> {
            "Felipe Gavilán",
            "Felipe Díaz",
            "Claudia Rodríguez",
            "Cluadia Celeste"
        };

        await Task.Delay(1000);
        foundPeople = people.Where(p => p.Contains(searchText)).ToList();
    }
}
```

EventCallback

En oportunidades no queremos que se ejecute una acción en un mismo componente sino que queremos que sea el componente padre el que realice tal acción. Para ello, se define un callback que permita elevar un evento hacia el componente padre.

MovieItem.razor

```

<div>
    <p>Title: <b><i>@Movie.Title</i></b>
        @if(DateTime.Today.Subtract(Movie.ReleaseDate).Days <= 31*3) {
            <span style="color:red"> (New!)</span>
        }
    </p>
    <p>Release date: <b>@Movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
    @if(ShowButtons) {
        <div>
            <button @onclick="DeleteMovie">Delete</button>
        </div>
    }
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback DeleteMovie { get; set; }
}
    
```

MoviesList.razor

```

@if(Movies is null) {
    
}
else if(Movies.Count == 0) {
    <p>No movies to show</p>
}
else {
    <input type="checkbox" @bind="ShowButtons" />
    <span>Show buttons</span>
    @foreach(var movie in Movies) {
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    }
}

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }

    bool ShowButtons = false;

    private void DeleteMovie() {
    }
}
    
```

```

        Console.WriteLine("Deleting a movie");
    }
}

```

Con esto se conseguiría un evento demasiado básico ya que no tenemos el nombre de la película que se borraría. Para tener un evento más completo que nos indique la película a borrar:

MovieItem.razor

```

<div>
    <p>Title: <b><i>@Movie.Title</i></b>
    @if(DateTime.Today.Subtract(Movie.ReleaseDate).Days <= 31*3) {
        <span style="color:red"> (New!)</span>
    }
    </p>
    <p>Release date: <b>@Movie.ReleaseDate.ToString("dd MM yyyy")</b></p>
    @if>ShowButtons {
        <div>
            <button @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
        </div>
    }
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }
}

```

MoviesList.razor

```

@if(Movies is null) {
    
}
else if(Movies.Count == 0) {
    <p>No movies to show</p>
}
else {
    <input type="checkbox" @bind="ShowButtons" />
    <span>Show buttons</span>
    @foreach(var movie in Movies) {
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    }
}

```

```
@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }

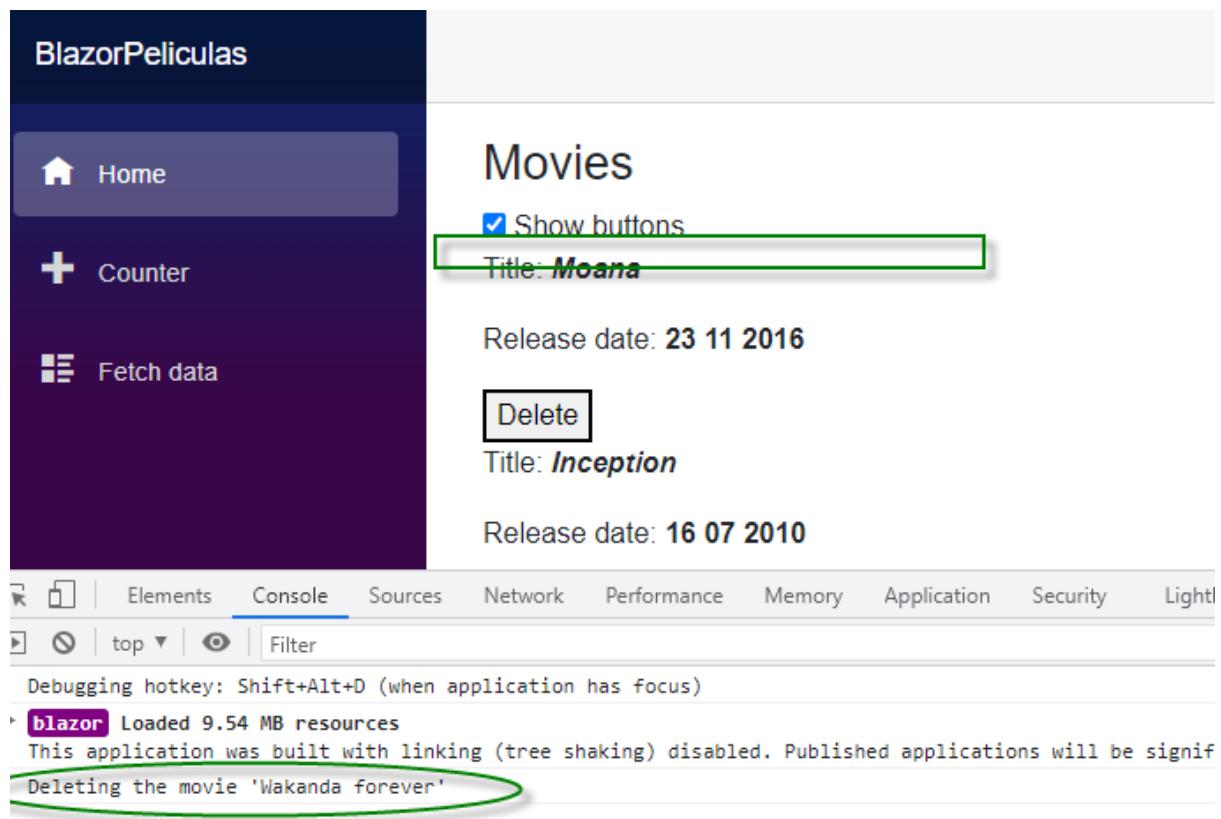
    bool ShowButtons = false;

    private void DeleteMovie(Movie movie) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}
```

Con el `!` que sigue a `Movies` conseguimos eliminar el warning que nos avisa que el objeto podría ser nulo. Esto lo hacemos porque sabemos que nunca podría ser nulo ya que es un parámetro obligatorio.

The screenshot shows a Blazor application titled "BlazorPelículas". On the left, there's a navigation menu with "Home", "Counter", and "Fetch data" options. The main content area is titled "Movies". It displays a list of movies with their titles and release dates. The first movie is "Wakanda forever (New!)". Below it is another movie entry with "Delete" and "Moana" highlighted. At the bottom of the screen, a developer tools console is open, showing the following output:

```
Debugging hotkey: Shift+Alt+D (when application has focus)
▶ blazor Loaded 9.54 MB resources
This application was built with linking (tree shaking) disabled. Published applications will be signific
```



Vemos que al clickear el botón de la primera imagen, desapareció la película en cuestión y se mostró el mensaje en la consola.

RenderFragment

Permite pasar a otro componente contenido HTML. Cada fragmento que se desea pasar al componente hijo se le puede poner un nombre y después definir el contenido de cada uno en el componente padre. A su vez, en el componente hijo se puede definir un contenido por defecto para el caso en el que no sea definido en el componente padre:

```

MoviesList.razor
@if(Movies is null) {
    @if(Loading is null) {
        
    }
    else {
        @Loading
    }
}
else if(Movies.Count == 0) {
    @if(NoRecords is null) {
        <p>No movies to show</p>
    }
}

```

```
        }
    else {
        @NoRecords
    }
}
else {
    <input type="checkbox" @bind="ShowButtons" />
    <span>Show buttons</span>
    @foreach(var movie in Movies) {
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    }
}

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

    bool ShowButtons = true;
    private void DeleteMovie(Movie movie) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}
```

Index.razor

```
@page "/"

<div>
    <h3>Movies </h3>
    <div>
        <MoviesList Movies="GetMovies()">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    List<Movie> GetMovies() {
```

```
return new List<Movie>() {
    new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
    new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
    new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
};
```

RenderFragment genérico

El limitante anterior es que el contenido que se envía es bastante genérico. En el ejemplo anterior, por ejemplo, tanto el mostrado del mensaje de cargando como el de no hay películas es un comportamiento básico de cualquier lista. Sería genial poder tener un componente genérico que muestre el cargando mientras que el objeto X se esté cargando y el mensaje de que no hay registros cuando el Count = 0.

Para ello, se procede con el código siguiente. Vale aclarar que en el componente padre no existe la variable movie dentro del componente HasRecords por lo que se debe usar **context**. Si no se desea usar ese nombre, se le debe pasar al parámetro **Context** la variable deseada.

GenericList.razor

```
@typeparam TItem
@if (List is null) {
    @if (Loading is null) {
        
    }
    else {
        @Loading
    }
}
else if (List.Count == 0) {
    @if (NoRecords is null) {
        <p>No movies to show</p>
    }
    else {
        @NoRecords
    }
}
else {
```

```

@foreach (var elem in List) {
    @HasRecords(elem)
}
}

@code {
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;
    [Parameter]
    public RenderFragment<TItem> HasRecords { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public List<TItem> List { get; set; } = null!;
}

```

Es muy importante la sentencia `@typeparam` para indicar el nombre de la variable genérica que se usará. Esto es porque la Lista no tiene que ser de películas ya que puede ser de usuarios, facturas, pedidos, etc. En tal caso, se debe usar un nombre genérico para no atar nuestro componente a una clase (Movie) específica.

MoviesList.razor

```

<input type="checkbox" @bind="ShowButtons" />
<span>Show buttons</span>
<GenericList List="Movies">
    <Loading>
        @Loading
    </Loading>
    <NoRecords>
        @NoRecords
    </NoRecords>
    <HasRecords Context="movie">
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    </HasRecords>
</GenericList>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;
}

```

```
bool ShowButtons = true;
private void DeleteMovie(Movie movie) {
    Console.WriteLine($"Deleting the movie '{movie.Title}'");
    Movies!.Remove(movie);
}
```

En nuestro MoviesList pasamos los RenderFragment de **Loading** y **NoRecords** al **GenericList**. También el HasRecords con el Context de la película en cuestión.

Ciclo de vida de un componente

OnInitialized / OnInitializedAsync

Sirven para inicializar un componente. En esta etapa de inicialización podríamos disparar, por ejemplo, una petición HTTP hacia el servidor para obtener los datos a mostrar. La diferencia entre estos dos métodos es que el primero es síncrono y el otro es asíncrono.

Así, por ejemplo, si en la etapa de inicialización tu quieras realizar una petición HTTP hacia un servidor, debes de utilizar un **OnInitializedAsync**. Sin embargo, si tu lo que quieres hacer es una acción que es síncrona, por ejemplo validar un parámetro, pues en ese caso basta con que utilices **OnInitialized**.

OnParametersSet / OnParametersSetAsync

Estos son ejecutados cuando el componente ha recibido todos los parámetros y sus valores han sido asignados a sus respectivas propiedades.

Aunque esto de tener los parámetros asignados ocurre también en la etapa de inicialización, los métodos de **OnParametersSet** se ejecutan cada vez que se actualizan los parámetros. A diferencia de los métodos de **OnInitialized** que se ejecutan una única vez.

OnAfterRender / OnAfterRenderAsync

Se ejecutan cuando el componente ha finalizado de renderizar. Es decir, cuando el HTML ya se encuentra mostrado en pantalla. Esta es la etapa ideal para realizar operaciones sobre estos elementos HTML.

Por ejemplo, utilizar una librería de JavaScript para trabajar sobre alguno de los elementos HTML. **Estos métodos se ejecutan cada vez que el componente se renderiza.**

ShouldRender

Podemos utilizarlo para indicar si queremos que un componente realice renderización posteriores a su renderización inicial.

Vamos entonces a utilizar los distintos métodos del ciclo de vida de un componente. En nuestro caso vamos a utilizar la versión síncrona, pero todo lo que aprendamos aplica para la versión asíncrona también.

MoviesList.razor

```
<input type="checkbox" @bind="ShowButtons" />
Show buttons
<GenericList List="Movies">
    <Loading>
        @Loading
    </Loading>
    <NoRecords>
        @NoRecords
    </NoRecords>
    <HasRecords Context="movie">
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    </HasRecords>
</GenericList>

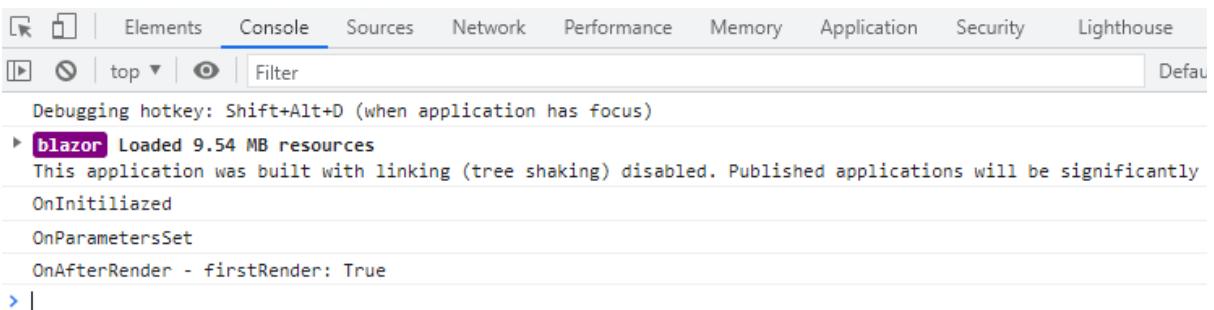
@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;
    protected override void OnInitialized() {
        Console.WriteLine("OnInitialized");
    }

    protected override void OnParametersSet() {
        Console.WriteLine("OnParametersSet");
    }

    protected override void OnAfterRender(bool firstRender) {
        Console.WriteLine($"OnAfterRender - firstRender: {firstRender}");
    }

    protected override bool ShouldRender() {
        Console.WriteLine("ShouldRender");
        return true;
    }

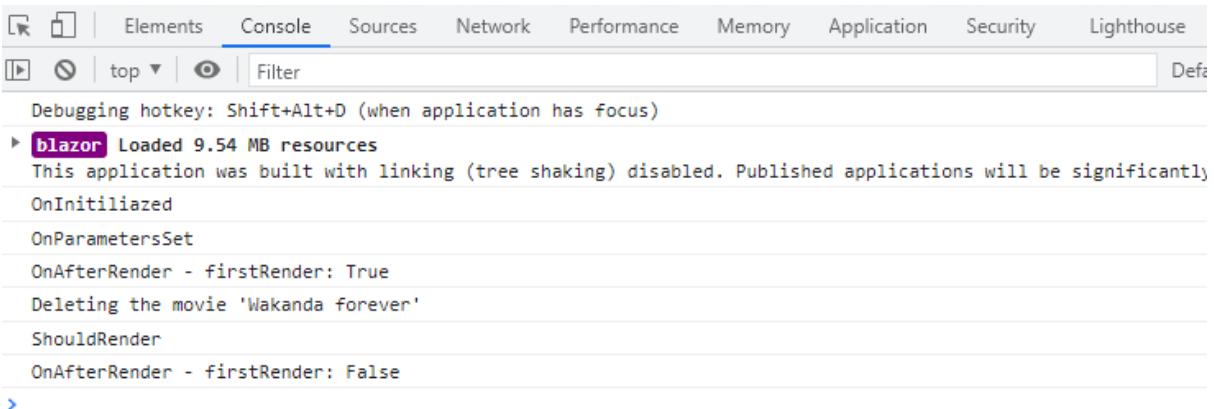
    bool ShowButtons = true;
    private void DeleteMovie(Movie movie) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}
```



```
Debugging hotkey: Shift+Alt+D (when application has focus)
▶ blazor Loaded 9.54 MB resources
This application was built with linking (tree shaking) disabled. Published applications will be significantly smaller.
  OnInitialized
  OnParametersSet
  OnAfterRender - firstRender: True
> |
```

El primero es **OnInitialized**, luego **OnParametersSet** porque tenemos parámetros y después **OnAfterRender**.

Hacemos click en borrar Wakanda:



```
Debugging hotkey: Shift+Alt+D (when application has focus)
▶ blazor Loaded 9.54 MB resources
This application was built with linking (tree shaking) disabled. Published applications will be significantly smaller.
  OnInitialized
  OnParametersSet
  OnAfterRender - firstRender: True
  Deleting the movie 'Wakanda forever'
  ShouldRender
  OnAfterRender - firstRender: False
> |
```

Vemos que logueo el borrado de la misma, pregunta si debe volver a renderizar (actualizar su parte visual) el componente y luego se ejecuta nuevamente el **OnAfterRender** pero dice que no es la primera vez (es lógico porque ya se había renderizado antes al ser cargado). Vemos que no se vuelve a ejecutar **OnParameterSet** porque el listado de películas no fue modificado desde fuera del componente. El botón Borrar del componente **Movieltem** ejecutó el evento y fue **MoviesList** quién eliminó la película de la lista.

Si, por temas de rendimiento, nuestro componente se encuentra realizando muchos cambios podríamos necesitar que no se renderice el componente con cada cambio realizado automáticamente. Con **ShouldRender** podemos contestar **false** y evitar que se actualice visualmente permanentemente.

Inyección de dependencias

La inyección de dependencias es un mecanismo el cual nos permite suplir las dependencias de una clase desde otra. Típicamente, esta inyección de dependencias la hacemos a través del constructor de la clase, aunque también podemos hacerlo a través de una propiedad.

En ASP.NET Core tenemos una manera sencilla de utilizar inyección de dependencias en componentes y clases. Esas configuraciones las hacemos a través de la clase Startup en el método Configure Services. Aquí podemos configurar nuestros servicios.

Cuando hablamos de servicios nos referimos a las clases o interfaces, las cuales podemos utilizar a través de la inyección de dependencias.

Es decir, que a través de la inyección de dependencias podemos solicitar una instancia de una clase. En el caso de los componentes, podemos utilizar a la directiva **@inject** para injectar una dependencia. La razón de utilizar la inyección de dependencias es para poder centralizar el mecanismo que configura nuestras dependencias en un solo lugar. Esto nos permite tener una arquitectura de software más flexible.

Por defecto, en Blazor se nos dan tres servicios que podemos utilizar a través de la inyección de dependencias

[httpClient](#)

Sirve para realizar peticiones HTTP hacia un servidor web.

[IJSRuntime](#)

Nos permite realizar operaciones con JavaScript.

[Navigation Manager](#)

nos permite trabajar con la navegación del usuario desde nuestro código.

Estos tres servicios no tenemos que configurarlos, son servicios por defecto que el framework de Blazor nos ofrece.

[Tiempo de vida de servicios](#)

Normalmente existen tres tiempos de vida para un servicio:

[Scoped](#)

Significa que el servicio va a vivir dentro de un contexto determinado. En ambientes web este contexto es una petición http. Sin embargo, en Blazor del lado del cliente no trabajamos en el contexto de una petición http. Dado que el código se encuentra en el navegador del usuario. Por lo que Scoped se comporta como un singleton en Blazor del lado del cliente.

Sin embargo, en el lado del servidor si tenemos contextos http.

[Singleton](#)

En este caso se crea una única instancia de un servicio y esta misma instancia es utilizada por siempre. Es decir, que si tenemos una clase a la cual servimos a través de la inyección de dependencias, si desde un componente yo solicito esa clase y está configurada como un singleton desde otro componente, yo puedo solicitar esa misma clase y se me va a dar el mismo objeto, es decir, la misma instancia de la

clase. No trabajaremos con objetos diferentes, sino que será el mismo objeto a lo largo de tu aplicación.

Transient

Significa transitorio. En ésta se crean distintas instancias del servicio cada vez que dicho servicio solicitado. Es decir, volviendo al ejemplo anterior, si solicitamos una clase en un componente y ésta es Transient y luego solicitamos la misma clase pero en otro componente, se le van a dar dos objetos diferentes a cada componente, porque al ser transitorio lo que quiere decir es que se nos dan instancias distintas de la clase.

Crearemos una clase Service.cs que tendrá 2 clases: **SingletonService** y **TransientService**.

Recordamos que estamos en el contexto de una aplicación de las pruebas en bloque en donde este código que se encuentra en el proyecto Client va a ejecutarse dentro del navegador del usuario. Entonces, lo que vamos a ver es que el singleton se va a mantener vivo. El valor que coloquemos aquí se va a mantener vigente para la sesión del usuario y el transient va a ser diferente siempre y cuando el usuario salga y entre de un componente. Es decir, para cada instancia del servicio Transient vamos a tener un valor distinto.

Esto es diferente a lo que tendríamos si estuviéramos en el proyecto Server. En el proyecto Server ahí estamos en un web API que está centralizado en un servidor y por tanto en el caso del proyecto Server, un singleton no sería de la sesión del usuario, sino que sería básicamente para todos los usuarios. ¿Por qué?

Porque recordamos que el servidor es único y por tanto sería la misma instancia para todos los usuarios distintos que consulten a ese servidor, mientras que el proyecto Client se descarga en el navegador del usuario y por tanto, el singleton va a ser de esa sesión del navegador del usuario, de esa sesión de la pestaña del navegador del usuario.

Los servicios se configuran en **Program.cs**. Se pueden agregar directamente a builder.services o crear una función y pasar el builder.services como parámetro para centralizar en una función todos los servicios que se agregarían.

Services.cs

```
namespace BlazorPeliculas.Client {
    public class SingletonService {
        public int Value { get; set; }
    }

    public class TrasientService {
        public int Value { get; set; }
    }
}
```

Program.cs

```
using BlazorPeliculas.Client;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

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

/*
builder.Services.AddSingleton<SingletonService>();
builder.Services.AddTransient<TransientService>();
*/
configureServices(builder.Services);
await builder.Build().RunAsync();

void configureServices(IServiceCollection services) {
    services.AddSingleton<SingletonService>();
    services.AddTransient<TransientService>();
}
```

Counter.razor

```
@page "/counter"
@inject SingletonService singleton
@inject TransientService transient

<PageTitle>Counter</PageTitle>

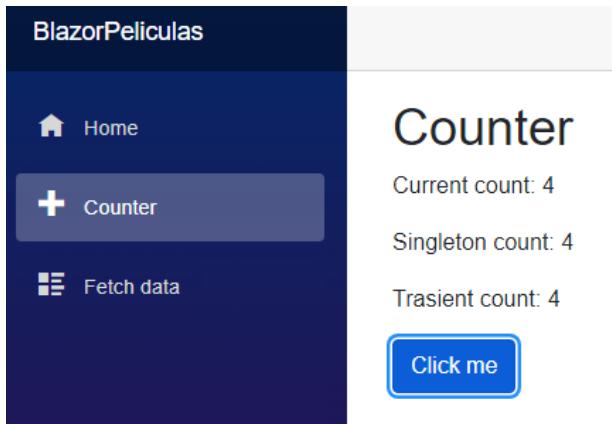
<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>
<p role="status">Singleton count: @singleton.Value</p>
<p role="status">Transient count: @transient.Value</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
        singleton.Value = currentCount;
        transient.Value = currentCount;
    }
}
```



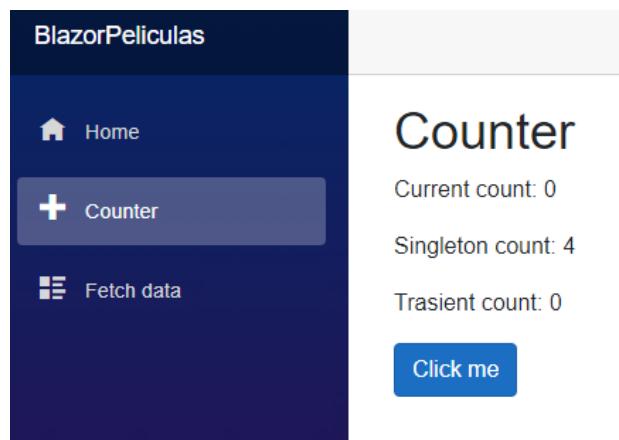
que el Trasient se instanciará una nueva copia cada vez que sea solicitada.

La sesión del Singleton está atada a la pestaña en cuestión. Si duplicara la pestaña cada pestaña manejaría su Singleton de manera independiente y cada uno mantendría su valor aún si fuéramos a Home y volviéramos.

El objeto Singleton es, como dijimos, para toda la sesión. Si inyectáramos el singleton en Home e imprimiéramos su valor, veríamos que tanto en Counter como en Home se mostraría siempre el mismo valor.

Ejecutamos el código y vamos al tab de **Counter** y presionamos el botón, los 3 valores se actualizan oportunamente con cada click.

Si nos vamos a Home y volvemos, vemos que el valor de Singleton sigue inalterado porque el objeto es la única copia que se utilizaría en toda la vida de la sesión del navegador mientras



Servicios con interfaces

Ahora que ya sabemos un poco más acerca de la inyección de dependencias, lo que vamos a hacer es que vamos a aprender a usarla, pero con interfaces. La idea de hacer las cosas de esta manera es tener una aplicación más flexible para que yo pueda en el futuro cambiar la implementación de un servicio.

Por ejemplo, actualmente, el listado de películas está hardcodeado en el **Index.razor**. En el futuro, esa información saldrá de una base de datos, pero cuando eso suceda no sería correcto tener que modificar dicho componente. Lo correcto sería que **Index.razor** consumiera un servicio que le devuelva el listado. Si ese servicio tomara los datos de un archivo de texto, una base de datos o mismo estuviera hardcodeado en el código el **Index.razor** debería seguir inalterable aún si cambiáramos el origen de los datos (texto, BD o hardcodeado). Esto es así porque la responsabilidad de **Index.razor** es mostrar el listado de películas y no tener el listado del mismo.

Entonces, lo que vamos a hacer es que no solamente vamos a centralizar esto en un servicio, sino que lo vamos a colocar detrás de una interfaz para que, como veremos en un momento en el futuro, la implementación de ese servicio puede cambiar y esto no va a afectar en nada a cualquier componente o clase en general que utilice este listado de películas.

Creamos una nueva carpeta **Repositories** y en ella agregamos una nueva clase del tipo **Interface** llamada **IRepository.cs**. Una interfaz es, básicamente, un conjunto de "firmas" que una clase tiene que implementar, es decir, una interfaz es como un contrato el cual ciertas clases tienen que cumplir. En el agregamos el método **GetMovies()** que devuelve un **List<Movie>**. Luego agregamos una clase **Repository.cs** que implementa **IRepository**. Al hacer eso, VS nos muestra un mensaje de error y la forma de solucionarlo propuesta es agregar las implementaciones que faltan.

IRepository.cs

```
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public interface IRepository {
        List<Movie> GetMovies();
    }
}
```

Repository.cs

```
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public class Repository : IRepository {
        public List<Movie> GetMovies() {
            return new List<Movie>() {
                new Movie {Title = "Wakanda forever", ReleaseDate = new DateTime(2022, 11, 11)},
                new Movie {Title = "Moana", ReleaseDate = new DateTime(2016, 11, 23)},
                new Movie {Title = "Inception", ReleaseDate = new DateTime(2010, 7, 16)}
            };
        }
    }
}
```

Agregamos el servicio que usaremos en **Program.cs**.

Program.cs

```
using BlazorPeliculas.Client;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
```

```
builder.RootComponents.Add<HeadOutlet>("head::after");

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

configureServices(builder.Services);
await builder.Build().RunAsync();

void configureServices(IServiceCollection services) {
    services.AddSingleton<SingletonService>();
    services.AddTransient<TransientService>();
    services.AddSingleton< IRepository, Repository>(); /* Inyectar un IRepository
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
    clase Repository. */
}
```

Agregamos el objeto como singleton para que si yo borro una película que esté en memoria, se va a quedar borrada, al menos en la sesión de esa pestaña. En nuestro componente ([Index.razor](#)) inyectaremos un IRepository pero en tiempo de ejecución tendremos una instancia de Repository.

Como sabemos que los repositorios serán utilizados a lo largo de todo nuestro proceso y para no tener que incluir todo el path al hacer el inject, agregamos el using correspondiente en [_Imports.razor](#).

```
_Imports.razor
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorPeliculas.Client
@using BlazorPeliculas.Client.Shared
@using BlazorPeliculas.Client.Utilities
@using BlazorPeliculas.Shared.Entities
@using BlazorPeliculas.Client.Repositories
```

Ahora podríamos hacer el inject de IRepository que, en tiempo de ejecución, tendríamos una copia de la clase Repository que tiene el listado de películas (con la función GetMovies).

Pero si mañana, si mañana yo me invento otra clase, digamos RealRepository, la cual también va a implementar IRepository, entonces con el solo hecho de yo venir

hacia la clase **Program.cs** y decir por ejemplo RealRepository (services.AddSingleton< IRepository, RealRepository>()), pues ya ahí se va a utilizar esa implementación de verdad de mi IRepository. Y este cambio tan sencillo que yo estoy haciendo aquí se va a propagar en toda mi aplicación y va a aplicar a cualquier componente o clase que utilice el IRepository. Estamos centralizando de esta manera qué implementación se va a utilizar para el IRepository, sin importar que estemos utilizando el IRepository en 100 componentes. Con cambiar esto en una línea, pues se va a hacer ese cambio en los 100 componentes.

Esto tiene un nombre, esto es uno de los cinco principios **SOLID** y se llama el principio de inversión de dependencias en donde nuestras clases o en este caso nuestro componente, no depende de un tipo concreto como el **Repository**, sino que depende de un tipo abstracto como el **IRepository**, es decir, depende de una abstracción. Y de esta manera yo tengo flexibilidad para cambiar en tiempo de ejecución. Si vamos a utilizar Repository o RealRepository o SQLRepository o OracleRepository o lo que yo quiera.

Es decir, yo tengo mucha flexibilidad a la hora de decidir en tiempo de ejecución cuál va a ser la implementación que vamos a utilizar.

Index.razor

```
@page "/"
@inject IRepository repository

<div>
    <h3>Movies</h3>
    <div>
        <MoviesList Movies="Movies">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie> ? Movies{ get; set; }
    protected override void OnInitialized()
    {
        Movies = repository.GetMovies();
    }
}
```

Cabecera del HTML

PageTitle

Esta etiqueta nos permite cambiar el título de la página

Counter.razor

```
@page "/counter"
@inject SingletonService singleton
@inject TrasientService trasient

<PageTitle>Counter</PageTitle>

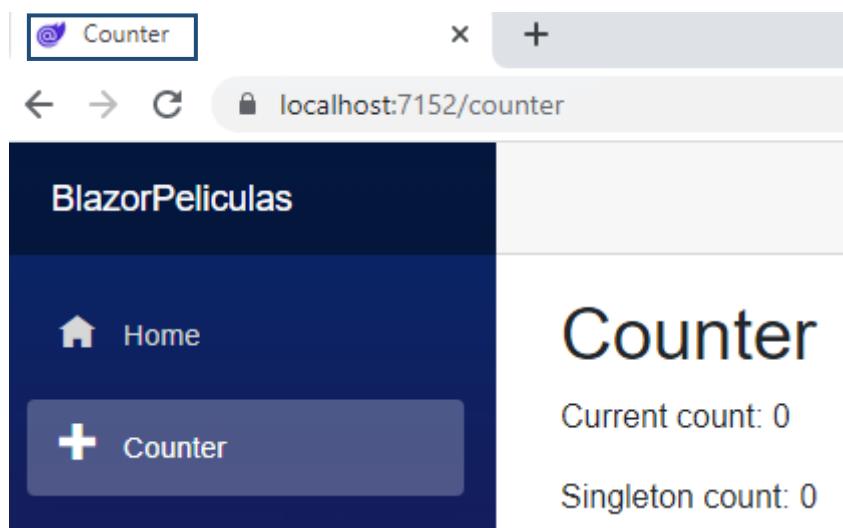
<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>
<p role="status">Singleton count: @singleton.Value</p>
<p role="status">Trasient count: @trasient.Value</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        currentCount++;
        singleton.Value = currentCount;
        trasient.Value = currentCount;
    }
}
```



HeadContent

Me permite agregar metatags.

Index.razor

```
@page "/"
@inject IRepository repository

<PageTitle>Blazor Movies</PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies" />
</HeadContent>

<div>
    <h3>Movies</h3>
    <div>
        <MoviesList Movies="Movies">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie> ? Movies{ get; set; }
    protected override void OnInitialized() {
        Movies = repository.GetMovies();
    }
}
```

El secreto está en **Program.cs** que modifica el contenido de **Index.html**.

En **Program.cs** están los Add en el **RootComponents** para agregar **#app** (App.razor, lo veremos más adelante) y **head::after**. El conte-

nido agregado con MetaContent se adiciona en el **head** de **Index.html**. Tanto **PageTitle** como **MetaContent** son dinámicos. Si ponemos una variable en ellos, el título o meta se actualizarán convenientemente.

Clases parciales

Para quiénes no les guste tener el código C# junto con el HTML, Blazor permite tener 2 archivos: uno para el código C# y otro para el HTML. Para ello, hay que crear una clase con el mismo nombre pero con extensión cs (por ejemplo: **Counter.razor.cs**). Al hacerlo, dicho nuevo archivo quedará “colgando” de **Counter.razor**. Para que pueda ser compilado, la clase debe ser una **partial class**.

```
Counter.razor
```

```
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>
<p role="status">Singleton count: @singleton.Value</p>
<p role="status">Transient count: @trasient.Value</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
```

```
Counter.razor.cs
```

```
using Microsoft.AspNetCore.Components;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] SingletonService singleton { get; set; } = null!;
        [Inject] TrasientService trasient { get; set; } = null!;

        private int currentCount = 0;

        private void IncrementCount() {
            currentCount++;
            singleton.Value = currentCount;
            trasient.Value = currentCount;
        }
    }
}
```

Los inject se realizan con el atributo **Inject** que está definido en el namespace **Microsoft.AspNetCore.Components**.

Layout

Cuando tenemos una aplicación web, es probable que tengamos elementos comunes que queremos que se visualicen en todos los componentes.

Por ejemplo, el menú a la izquierda es un elemento común de todas las páginas de mi aplicación.

¿Cómo logramos eso?

Pues utilizando un layout en la carpeta Shared. Es el componente **MainLayout.razor**. En él se ve el componente **NavMenu** que es el menú lateral.

Componente a renderizar

En el **@Body** que significa cuerpo, es donde se coloca el cuerpo del componente que estamos realizando, es decir, es donde se coloca el componente en sí que estamos renderizado, ya sea **Counter**, **Home**, **Fetch data**.

MainLayout.razor

```
@inherits LayoutComponentBase

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>
```

NavMenu.razor

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">BlazorPelículas</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
```

```

<div class="nav-item px-3">
    <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
        <span class="oi oi-home" aria-hidden="true"></span> Home
    </NavLink>
</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="counter">
        <span class="oi oi-plus" aria-hidden="true"></span> Counter
    </NavLink>
</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="fetchdata">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
    </NavLink>
</div>
</nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}

```

Definición del layout

Para definir el layout utilizamos un archivo que se llama **App.razor** (lo veremos más a fondo más adelante). Si la página a renderizar fue encontrada (Found) vemos que tenemos **RouteView** o vista de la ruta y **DefaultLayout**.

Yo puedo tener varios layout, no hay ningún problema con eso, pero por defecto tenemos el **MainLayout** y este está siendo configurado a través de la pressure utilizando este atributo de default layout.

App.razor

```

<Router AppAssembly="@typeof(App).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
    <NotFound>
        <PageTitle>Not found</PageTitle>
        <LayoutView Layout="@typeof(MainLayout)">
            <p role="alert">Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>

```

Invocar Javascript desde C#

En ocasiones querremos correr funciones/métodos de JS. Si bien hay librerías de Blazor para hacer esto, en el ejemplo, invocaremos al **confirm** de JS para preguntar si está seguro de que se quiere borrar una película. Para eso, se usa la librería **IJSRuntime**. Es necesario cambiar la devolución de void a un **async Task** y se utiliza un await **js.InvokeAsync** para quedarse esperando la respuesta. El **InvokeAsync** debe recibir cuál es el tipo de dato de la respuesta. En este caso: **bool**.

MoviesList.razor

```
@inject IJSRuntime js
<input type="checkbox" @bind="ShowButtons" />
<span>Show buttons</span>
<GenericList List="Movies">
    <Loading>
        @Loading
    </Loading>
    <NoRecords>
        @NoRecords
    </NoRecords>
    <HasRecords Context="movie">
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    </HasRecords>
</GenericList>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

    protected override void OnInitialized() {
        Console.WriteLine($"OnInitialized - Movies.Count: {Movies!.Count}");
    }

    protected override void OnParametersSet() {
        Console.WriteLine($"OnParametersSet - Movies.Count: {Movies!.Count}");
    }

    protected override void OnAfterRender(bool firstRender) {
        Console.WriteLine($"OnAfterRender - firstRender: {firstRender}");
    }

    protected override bool ShouldRender()
```

```

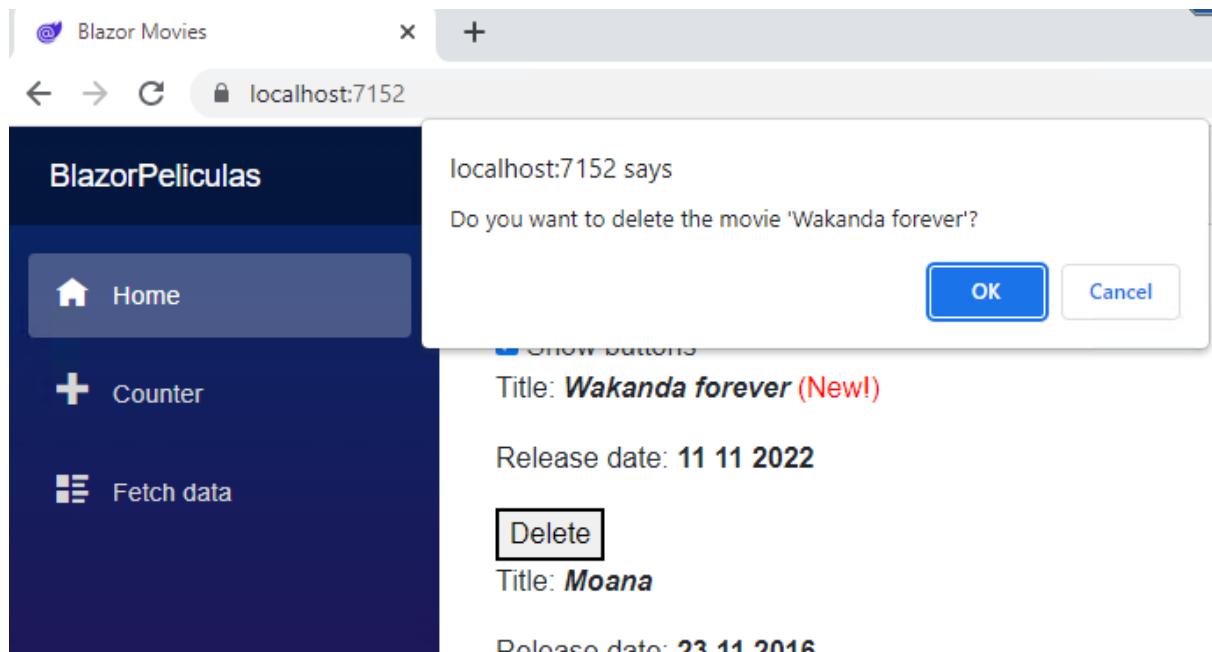
Console.WriteLine("ShouldRender");
return true;
}

bool ShowButtons = true;
private async Task DeleteMovie(Movie movie) {
    var confirmed = await js.InvokeAsync<bool>("confirm", $"Do you want to delete the movie '{movie.Title}'?");

    if (confirmed) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}

<Found Context="routeData">
    <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
    <FocusOnNavigate RouteData="@routeData" Selector="h1" />
</Found>
<NotFound>
    <PageTitle>Not found</PageTitle>
    <LayoutView Layout="@typeof(MainLayout)">
        <p role="alert">Sorry, there's nothing at this address.</p>
    </LayoutView>
</NotFound>
</Router>

```



También lo podemos centralizar en una clase Helper. Para ello agregamos la carpeta **Helpers** y en ella la clase **IJSRuntimeExtensionMethods.cs**. Agregamos **BlazorPeliculas.Client.Helpers** en **_Imports.razor**. Es importante que tanto la clase como la función sean estáticas.

IJSRuntimeExtensionMethods.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Helpers {
    public static class IJSRuntimeExtensionMethods {
        public static async ValueTask<bool> Confirm(this IJSRuntime js, string message) {
            await js.InvokeVoidAsync("console.log", $"Asking: {{message}}");
            return await js.InvokeAsync<bool>("confirm", message);
        }
    }
}
```

MoviesList.razor

```
@inject IJSRuntime js
<input type="checkbox" @bind="ShowButtons" />
<span>Show buttons</span>
<GenericList List="Movies">
    <Loading>
        @Loading
    </Loading>
    <NoRecords>
        @NoRecords
    </NoRecords>
    <HasRecords Context="movie">
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    </HasRecords>
</GenericList>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

    protected override void OnInitialized() {
        Console.WriteLine($"OnInitialized - Movies.Count: {Movies!.Count}");
    }

    protected override void OnParametersSet() {
        Console.WriteLine($"OnParametersSet - Movies.Count: {Movies!.Count}");
    }
}
```

```

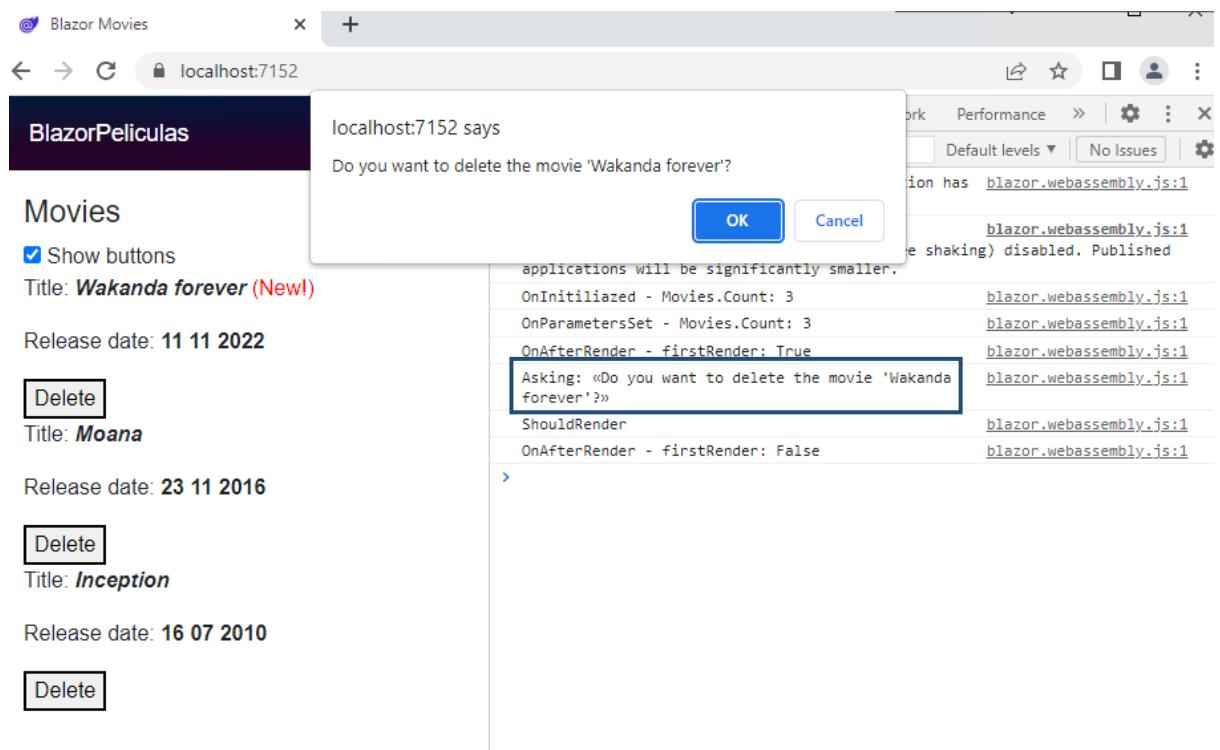
protected override void OnAfterRender(bool firstRender) {
    Console.WriteLine($"OnAfterRender - firstRender: {firstRender}");
}

protected override bool ShouldRender() {
    Console.WriteLine("ShouldRender");
    return true;
}

bool ShowButtons = true;
private async Task DeleteMovie(Movie movie) {
    var confirmed = await js.Confirm($"Do you want to delete the movie '{movie.Title}'?");

    if (confirmed) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}
}

```



Invocar C# desde Javascript

Función estática

Intentaremos obtener el valor de currentCount desde JS. Para ello, crearemos una función estática que devuelva el valor de una variable estática (que será copia de la variable original). Esto es porque la variable currentCount es un miembro de la instancia y la función es estática.

Primero, creamos una carpeta **js** en la carpeta **wwwroot**. Históricamente en la carpeta **wwwroot** ponemos todos los archivos estáticos. En ella agregamos un nuevo ítem del tipo javascript (buscarlo) y llamarlo **Utilities.js**. En él haremos una función que llamará el método `invokeMethodAsync` de DotNet que devuelve una promesa. Cuando ésta se cumpla, haremos un **console.log** con el valor que haya recibido en **result**.

Utilities.js

```
function DotNetStaticTest() {
    DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
        .then(result => {
            console.log('Count from js: ' + result);
        });
}
```

También es necesario agregar **Microsoft.JSInterop**, cambiar la función que devuelvía **void** por un **async Task**, hacer la llamada con el **js.InvokeAsync** de la función en DotNet y agregar el atributo **JSInvokable** a la función que podrá ser invocable desde JS.

Counter.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] SingletonService singleton { get; set; } = null!;
        [Inject] TrasientService trasient { get; set; } = null!;
        [Inject] IJSRuntime js { get; set; } = null!;

        private int currentCount = 0;
        private static int staticCurrentCount = 0;

        private async Task IncrementCount() {
            currentCount++;
            singleton.Value = currentCount;
            trasient.Value = currentCount;
            staticCurrentCount = currentCount;
            await js.InvokeVoidAsync("DotNetStaticTest");
        }
    }
}
```

```
}
```

```
[JSInvokable]
```

```
public static Task<int> GetCurrentCount() {
    return Task.FromResult(staticCurrentCount);
}
```

```
}
```

También es necesario agregar el js en el archivo [index.html](#).

index.html

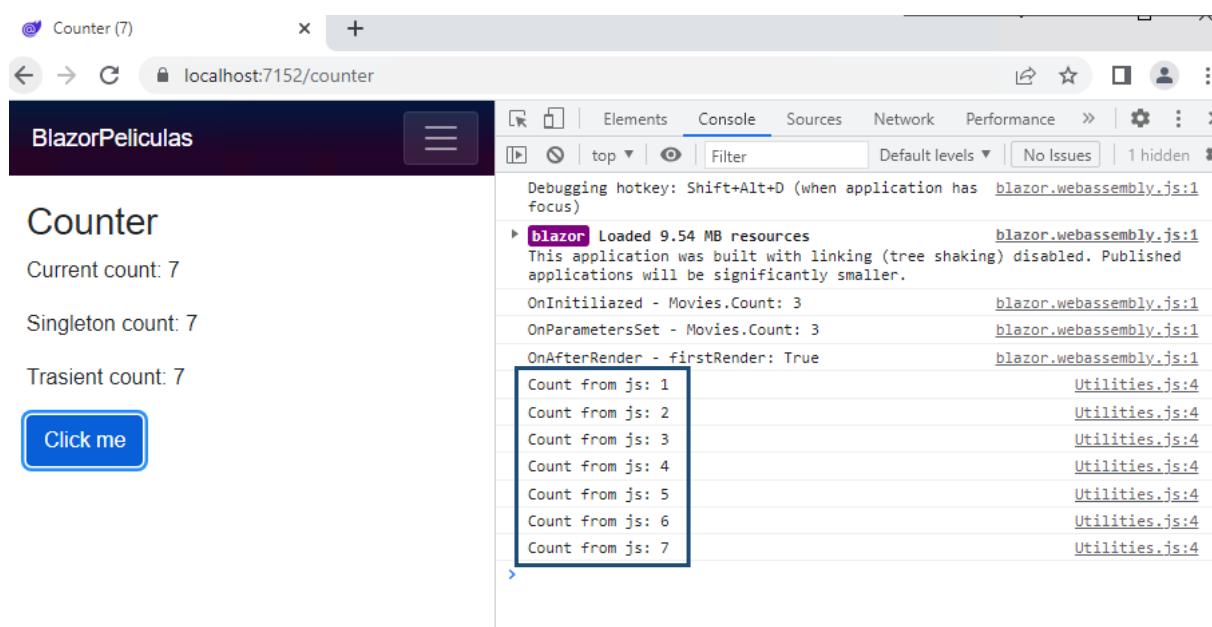
```
<!DOCTYPE html>
<html lang="en">

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

<body>
    <div id="app">
        <svg class="loading-progress">
            <circle r="40%" cx="50%" cy="50%" />
            <circle r="40%" cx="50%" cy="50%" />
        </svg>
        <div class="loading-progress-text"></div>
    </div>

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

</html>
```



Método de instancia desde JS

Primero que nada, le ponemos el atributo `JSInvokable` para que se pueda invocar y el cambiamos el scope a público.

Counter.razor.cs

```

using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPelículas.Client.Pages {
    public partial class Counter {
        [Inject] SingletonService singleton { get; set; } = null!;
        [Inject] TrasientService trasient { get; set; } = null!;
        [Inject] IJSRuntime js { get; set; } = null!;

        private int currentCount = 0;
        private static int staticCurrentCount = 0;

        [JSInvokable]
        public async Task IncrementCount() {
            currentCount++;
            singleton.Value = currentCount;
            trasient.Value = currentCount;
            staticCurrentCount = currentCount;
            await js.InvokeVoidAsync("DotNetStaticTest");
        }
    }
}

```

```
private async Task IncrementCountJs() {
    await js.InvokeVoidAsync("DotNetInstanceTest", DotNetObjectReference.Create(this));
}

[JSInvokable]
public static Task<int> GetCurrentCount() {
    return Task.FromResult(staticCurrentCount);
}
}
```

Creamos la función **DotNetInstanceTest** en el **Utilities.js** que reciba un **dotNetHelper** como parámetro que es una referencia de un objeto de C# que va a ser utilizable desde JavaScript. Como se vio en el código anterior, se crea una referencia a partir de **this** (la clase **Counter**) con el objeto **DotNetObjectReference**.

Utilities.js

```
function DotNetStaticTest() {
    DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
        .then(result => {
            console.log('Count from js: ' + result);
        });
}

function DotNetInstanceTest(dotNetHelper) {
    dotNetHelper.invokeMethodAsync("IncrementCount");
}
```

Agregamos un botón en la página de **Counter.razor** que llamará a la función **IncrementCountJs**.

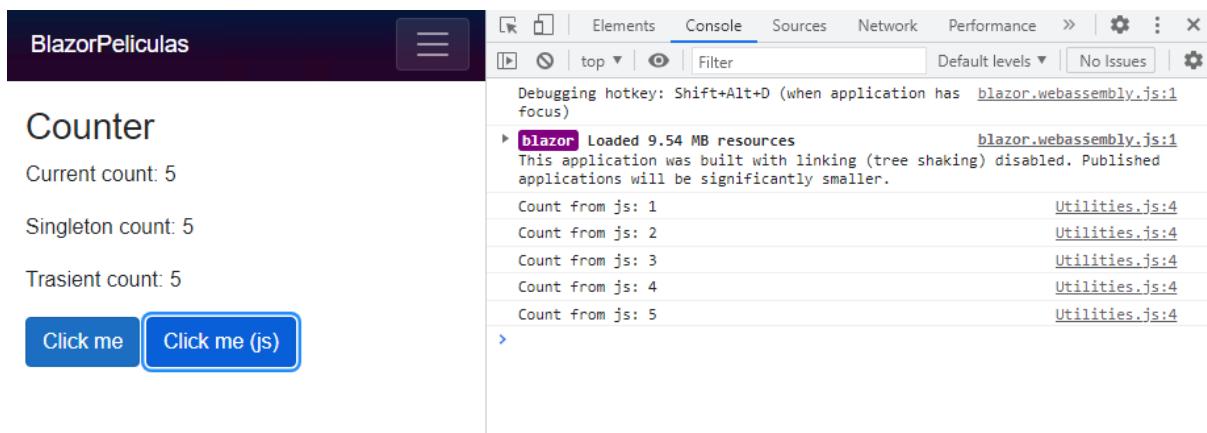
Counter.razor

```
@page "/counter"

<PageTitle>Counter (@currentCount)</PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies (@currentCount)" />
</HeadContent>
<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>
<p role="status">Singleton count: @singleton.Value</p>
<p role="status">Trasient count: @trasient.Value</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
<button class="btn btn-primary" @onclick="IncrementCountJs">Click me (js)</button>
```



¿Por qué vemos el “Count from js”? Porque el botón “Click me (js)” invoca a la función **IncrementCountJs** ésta invoca a **DotNetInstanceTest** y ésta invoca a **IncrementCount**. Este método invoca a **DotNetStaticTest** que es el que termina escribiendo en consola.

Aislamiento de JS

La idea es que tendremos muchos archivos JS y sólo invocar los JS que necesitamos en cada componente para que el cargado sea lo más rápido posible. Además, evitamos que algunos componentes no tengan acceso a código de JS que no deberían tener.

Para esto lo que hacemos es que creamos módulos de JavaScript a través de los cuales vamos a poder realizar una carga, guardar ese módulo en memoria en una variable de C# y a través de esa variable es que vamos a poder trabajar con las funciones exportadas de dicho módulo.

Por ejemplo, en la siguiente imagen, vemos que al entrar en el **Home** se está descargando el **Utilities.js** que utilizamos antes con la página **Counter**. En este caso, sería conveniente que dicho js sólo sea descargado si entráramos en **Counter**.

Name	Status	Type	Initiator	Size	Time	Waterfall
localhost	304	document	Other	46 B	3 ms	
Utilities.js	200	script	(index)	(memory ca...)	0 ms	
blazor.webassembly.js	304	script	(index)	22 B	9 ms	
aspnetcore-browser-refresh.js	200	script	(index)	12.1 kB	3 ms	
open-iconic-bootstrap.min.css	200	stylesheet	app.css	(memory ca...)	0 ms	
open-iconic.woff	200	font	open-iconic-bootstra...	(memory ca...)	0 ms	
blazor.boot.json	304	fetch	blazor.webassembly.js...	20 B	2 ms	
favicon.png	200	png	Other	(disk cache)	1 ms	

Crearemos un nuevo archivo de JavaScript y lo voy a configurar como un módulo para que no se cargue de una vez llamado **Counter.js**. En él agregamos una función que **export** para que el archivo se comporte como módulo. Veremos que podremos utilizarlo aún sin agregar el mismo en el [Index.html](#).

Counter.js

```
export function showAlert(message) {
    return alert(message);
}
```

Para conseguir esto creamos una variable **module** del tipo **IJSObjectReference** en **Counter.razor.cs**. Luego, en la función IncrementCount seteamos el valor correspondiente para que haga la descarga del archivo necesario. Vale destacar que invocaremos **showAlert** a través del módulo importado y no a través IJSRuntime porque me va a dar un error, porque esta función **showAlert** solamente se encuentra en este módulo de JavaScript que tenemos en **module**.

Counter.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] SingletonService singleton { get; set; } = null!;
        [Inject] TrasientService trasient { get; set; } = null!;
        [Inject] IJSRuntime js { get; set; } = null!;

        IJSObjectReference? module;

        private int currentCount = 0;
        private static int staticCurrentCount = 0;

        [JSInvokable]
        public async Task IncrementCount() {
            module = await js.InvokeAsync<IJSObjectReference>("import", "./js/Counter.js");
            await module.InvokeVoidAsync("showAlert", "hello world");

            currentCount++;
            singleton.Value = currentCount;
            trasient.Value = currentCount;
            staticCurrentCount = currentCount;
            await js.InvokeVoidAsync("DotNetStaticTest");
        }

        private async Task IncrementCountJs() {
            await js.InvokeVoidAsync("DotNetInstanceTest", DotNetObjectReference.Create(this));
        }

        [JSInvokable]
        public static Task<int> GetCurrentCount() {
            return Task.FromResult(staticCurrentCount);
        }
    }
}
```

En la siguiente imagen podemos ver que el archivo **Counter.js** no fue invocado:

Name	Status	Type	Initiator	Size	Time	Waterfall
counter	304	document	Other	104 B	36 ms	
Utilities.js	200	script	counter	(memory ca...)	0 ms	
blazor.webassembly.js	304	script	counter	22 B	5 ms	
aspnetcore-browser-refresh.js	200	script	counter	12.1 kB	3 ms	
open-iconic-bootstrap.min.css	200	stylesheet	app.css-Infinity	(memory ca...)	0 ms	
blazor.boot.json	304	fetch	blazor.webassembly.js...	20 B	4 ms	
open-iconic.woff	200	font	open-iconic-bootstra...	(memory ca...)	0 ms	
favicon.png	200	png	Other	(disk cache)	1 ms	
dotnet.7.0.0.njnilnrssx.js	304	script	blazor.webassembly.js...	52 B	9 ms	
blazor-hotreload.js	200	script	blazor.webassembly.js...	802 B	1 ms	
blazor-hotreload	204	fetch	blazor-hotreload.js!1	42 B	202 ms	
BlazorPelículas.Server/	101	websocket	aspnetcore-browserr...	0 B	Pending	

Limpiamos el listado para que quede más evidente y después presionamos "Click me". Veremos que el archivo es descargado y el alert se muestra en pantalla:

Name	Status	Type	Initiator	Queued at 0	Started at 0.39 ms	Resource Scheduling	DURATION
Counter.js	200	script	blazor.webassembly.js...			Queueing	0.39 ms

Sin importar que volvamos a clickear en el botón, el archivo no se vuelve a cargar porque ASP.NET Core Blazor ya sabe que está cargado.

Aislamiento de CSS

En ocasiones querremos que algunos estilos se apliquen sólo a un componente. Para ello, crearemos un archivo llamado **Counter.razor.css**. Con esto, conseguiremos que los estilos definidos en este archivo sólo funcionen para este componente. Vale aclarar, que los estilos definidos de esta forma tienen mayor precedencia que los estilos globalmente (en **wwwroot/css/app.css**). Para que esto funcione, es necesario hacer un Rebuild solution y refrescar la página con Ctrl+F5.

Referencias a Componentes

En el componente **index.razor** tenemos el listado de películas. Si quisiéramos poder invocar un método del listado de películas, necesitamos un campo en el cual guardaremos una referencia al componente listado películas.

Primero crearemos un método público que pueda ser accedido desde el componente padre. Entonces, crearemos el método **CleanMovies()** que será el método al cuál llamaremos desde nuestra referencia al componente propiamente dicho:

MoviesList.razor

```
@inject IJSRuntime js
<input type="checkbox" @bind="ShowButtons" />
<span>Show buttons</span>
<GenericList List="Movies">
    <Loading>
        @Loading
    </Loading>
    <NoRecords>
        @NoRecords
    </NoRecords>
    <HasRecords Context="movie">
        <MovieItem Movie="movie" ShowButtons="ShowButtons" DeleteMovie="DeleteMovie"/>
    </HasRecords>
</GenericList>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;
```

```
public void CleanMovies() {
    if(Movies is not null) {
        Movies.Clear();
    }
}

protected override void OnInitialized() {
    Console.WriteLine($"OnInitialized - Movies.Count: {Movies!.Count}");
}

protected override void OnParametersSet() {
    Console.WriteLine($"OnParametersSet - Movies.Count: {Movies!.Count}");
}

protected override void OnAfterRender(bool firstRender) {
    Console.WriteLine($"OnAfterRender - firstRender: {firstRender}");
}

protected override bool ShouldRender() {
    Console.WriteLine("ShouldRender");
    return true;
}

bool ShowButtons = true;
private async Task DeleteMovie(Movie movie) {
    var confirmed = await js.Confirm($"Do you want to delete the movie '{movie.Title}'?");

    if (confirmed) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}
}
```

Definimos una variable del tipo **MoviesList** llamado **moviesList**. A su vez, en la definición del objeto agregamos la etiqueta `@ref` para poder setearle a dicha variable el componente real. Además, agregamos un botón que llame a un método de este componente en el que se invoque al método público creado en el código anterior.

Index.cs

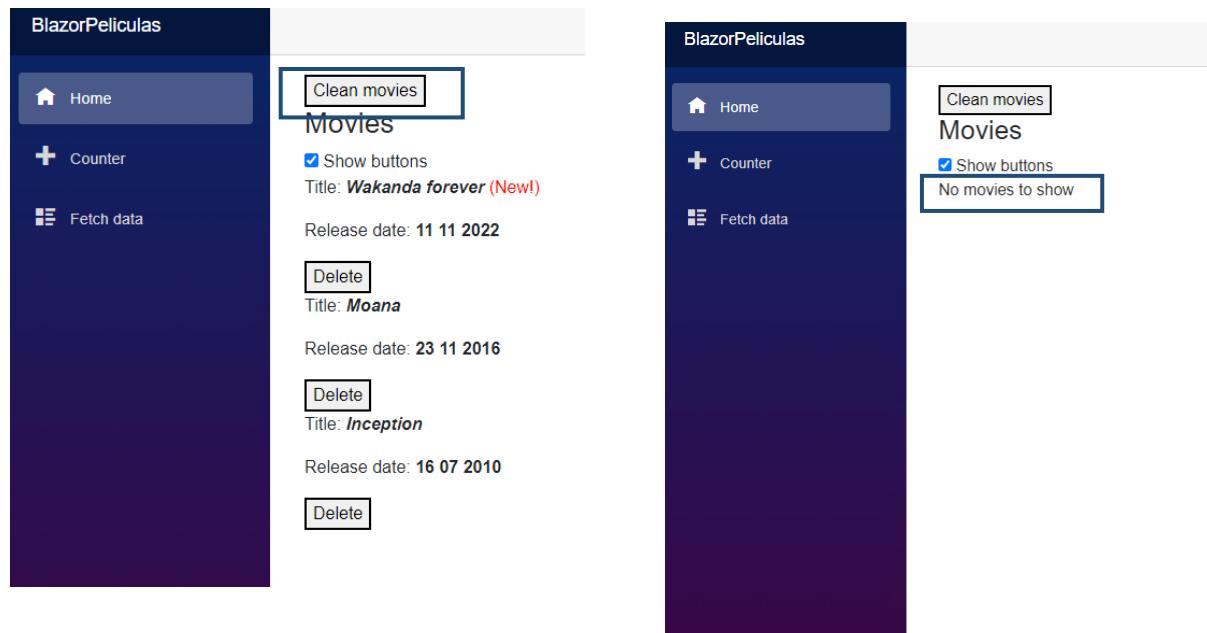
```
@page "/"
@inject IRepository repository

<PageTitle>Blazor Movies</PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies" />
</HeadContent>
```

```

<button @onclick="CleanMovies">Clean movies</button>
<div>
    <h3>Movies</h3>
    <div>
        <MoviesList @ref="moviesList" Movies="Movies">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
@code {
    public List<Movie> ? Movies{ get; set; }
    public MoviesList? moviesList;
    protected override void OnInitialized() {
        Movies = repository.GetMovies();
    }

    private void CleanMovies() {
        moviesList!.CleanMovies();
    }
}
    
```



En la imagen de arriba vemos el nuevo botón y a la derecha que la lista desaparece al clickearlo.

Parámetros de cascada

Sabemos que podemos pasarle parámetros a componentes. Por ejemplo, tenemos **MoviesList** en el componente **index**. Al componente **MoviesList** le paso el parámetro **Movies**, es decir, yo puedo pasarle valores a un componente.

Sin embargo, a veces queremos un poquito más de flexibilidad, porque en ocasiones vamos a necesitar no simplemente pasarle parámetros, por ejemplo, al **Movies**, sino pasarle parámetros a componentes internos del **MoviesList**.

Así, por ejemplo, si vamos a **MoviesList** y supongamos que queremos pasarle un parámetro de película individual, ¿cómo podríamos hacer eso desde **index**?

Pues ahora mismo con las herramientas que tenemos no podemos hacerlo, por lo cual tenemos que aprender una nueva herramienta para pasar parámetros, que es los parámetros de cascada.

Los parámetros de cascada nos van a permitir definir un conjunto de parámetros en un componente padre y que cualquier componente hijo o nieto o lo que sea va a poder acceder a esos parámetros.

Para esta prueba, comentaremos los estilos de **Counter.razor.css**. En **MainLayout.cs** definimos el **CascadingValue** y le seteamos el color rojo. Hacemos similar con **Size**:

```
MainLayout.cs
@inherits LayoutComponentBase

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>
        <article class="content px-4">
            <CascadingValue Value="@Color" Name="Color">
                <CascadingValue Value="@Size" Name="Size">
                    @Body
                </CascadingValue>
            </CascadingValue>
        </article>
    </main>
</div>

@code {
    private string Color = "red";
    private string Size = "12px";
}
```

En **Counter.razor.cs**, creamos una variable con el atributo CascadingParameter llamado **Color** y otra **Size** similar a la anterior.

Counter.razor.cs

```
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] SingletonService singleton { get; set; } = null!;
        [Inject] TrasientService trasient { get; set; } = null!;
        [Inject] IJSRuntime js { get; set; } = null!;
        [CascadingParameter(Name = "Color")] protected string Color { get; set; } = null!;
        [CascadingParameter(Name = "Size")] protected string Size { get; set; } = null!;
        IJSObjectReference? module;

        private int currentCount = 0;
        private static int staticCurrentCount = 0;

        [JSInvokable]
        public async Task IncrementCount() {
            module = await js.InvokeAsync<IJSObjectReference>("import", "./js/Counter.js");
            await module.InvokeVoidAsync("showAlert", "hello world");

            currentCount++;
            singleton.Value = currentCount;
            trasient.Value = currentCount;
            staticCurrentCount = currentCount;
            await js.InvokeVoidAsync("DotNetStaticTest");
        }

        private async Task IncrementCountJs() {
            await js.InvokeVoidAsync("DotNetInstanceTest", DotNetObjectReference.Create(this));
        }

        [JSInvokable]
        public static Task<int> GetCurrentCount() {
            return Task.FromResult(staticCurrentCount);
        }
    }
}
```

Finalmente, en **Counter.razor** utilizamos dicha variable para setearle el estilo al h1 y al primer p.

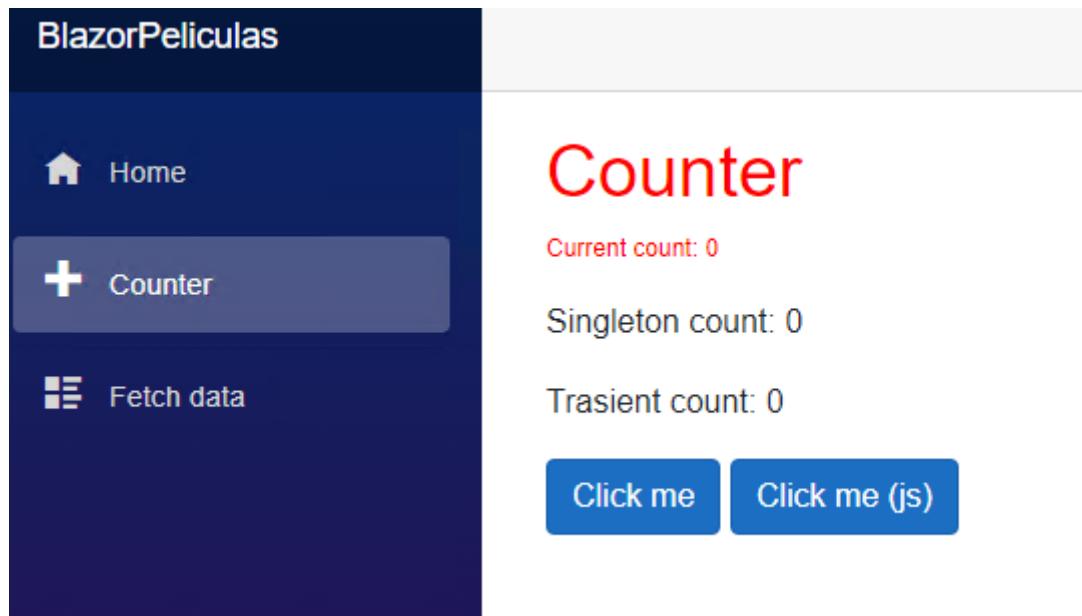
Counter.razor

```
@page "/counter"

<PageTitle> Counter (@currentCount) </PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies (@currentCount)" />
</HeadContent>
<h1 style="color:@Color">Counter </h1>

<p style="color:@Color;font-size:@Size" role="status">Current count: @currentCount </p>
<p role="status">Singleton count: @singleton.Value </p>
<p role="status">Trasient count: @trasient.Value </p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me </button>
<button class="btn btn-primary" @onclick="IncrementCountJs">Click me (js) </button>
```



Otra cosa que podemos hacer es utilizar una clase como el tipo de datos que vamos a pasar como parámetro.

Primero creamos una clase **AppState** en la carpeta **Helpers**:

AppState.cs

```
namespace BlazorPeliculas.Client.Helpers {
    public class AppState {
        public string Color { get; set; } = "green";
        public string Size { get; set; } = "14px";
```

{
}

Luego utilizamos instanciamos un objeto de dicha clase y la usamos para el CascadingValue:

MainLayout.cs

@inherits LayoutComponentBase

```
<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>

        <article class="content px-4">
            <CascadingValue Value="@appState">
                @Body
            </CascadingValue>
        </article>
    </main>
</div>

@code {
    AppState appState = new AppState();
}
```

Y realizamos los cambios necesarios para usar los datos de la appState que viene en el CascadingValue:

Counter.razor.cs

```
using BlazorPeliculas.Client.Helpers;
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] SingletonService singleton { get; set; } = null!;
        [Inject] TrasientService trasient { get; set; } = null!;
        [Inject] IJSRuntime js { get; set; } = null!;
        [CascadingParameter] protected AppState appState { get; set; } = null!;
        IJSObjectReference? module;
```

```
private int currentCount = 0;
private static int staticCurrentCount = 0;

[JSInvokable]
public async Task IncrementCount() {
    module = await js.InvokeAsync<IJSObjectReference>("import", "./js/Counter.js");
    await module.InvokeVoidAsync("showAlert", "hello world");

    currentCount++;
    singleton.Value = currentCount;
    trasient.Value = currentCount;
    staticCurrentCount = currentCount;
    await js.InvokeVoidAsync("DotNetStaticTest");
}

private async Task IncrementCountJs() {
    await js.InvokeVoidAsync("DotNetInstanceTest", DotNetObjectReference.Create(this));
}

[JSInvokable]
public static Task<int> GetCurrentCount() {
    return Task.FromResult(staticCurrentCount);
}
}
```

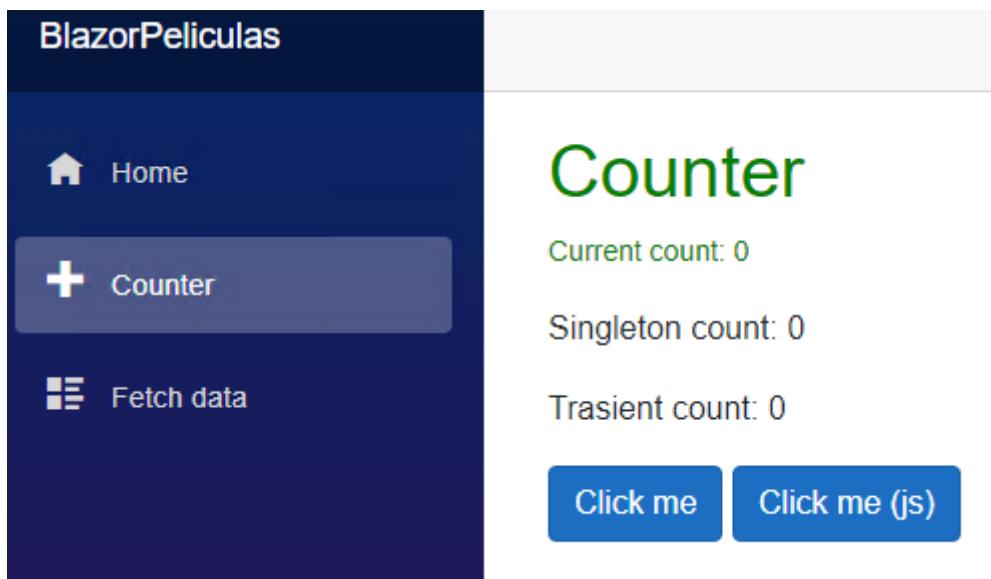
Counter.razor

```
@page "/counter"

<PageTitle>Counter (@currentCount) </PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies (@currentCount)" />
</HeadContent>
<h1 style="color:@appState.Color">Counter </h1>

<p style="color:@appState.Color;font-size:@appState.Size" role="status">Current count:
@currentCount</p>
<p role="status">Singleton count: @singleton.Value</p>
<p role="status">Trasient count: @trasient.Value</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me </button>
<button class="btn btn-primary" @onclick="IncrementCountJs">Click me (js) </button>
```



También se pueden cambiar los valores recibidos. Pondremos un select para poder cambiarle los valores en **Counter.razor**. Usaremos bind para que quede sincronizado.

Counter.razor

```
@page "/counter"
```

```
<PageTitle>Counter (@currentCount)</PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies (@currentCount)" />
</HeadContent>
<h1 style="color:@appState.Color">Counter</h1>

<div style="display:flex;" class="mt-4">
    <div style="width:150px;" class="me-2">
        <select @bind="@appState.Color" class="form-select">
            <option value="red">Red</option>
            <option value="green">Green</option>
            <option value="blue">Blue</option>
        </select>
    </div>
    <div style="width:150px;">
        <select @bind="@appState.Size" class="form-select">
            <option value="14px">14px</option>
            <option value="20px">20px</option>
            <option value="30px">30px</option>
        </select>
    </div>
</div>
<p style="color:@appState.Color;font-size:@appState.Size" role="status">Current count:<br/>@currentCount</p>
```

```
<p role="status">Singleton count: @singleton.Value</p>
<p role="status">Trasient count: @trasient.Value</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
<button class="btn btn-primary" @onclick="IncrementCountJs">Click me (js)</button>
```

Counter

Green 14px

Current count: 0

Singleton count: 0

Trasient count: 0

[Click me](#) [Click me \(js\)](#)

Counter

Blue 20px

Current count: 0

Singleton count: 0

Trasient count: 0

[Click me](#) [Click me \(js\)](#)

Counter

Red 30px

Current count: 0

Singleton count: 0

Trasient count: 0

[Click me](#) [Click me \(js\)](#)

Vemos así, que los cambios, afectan automáticamente a los estilos de los objetos HTML.





Ruteo

Nuevos códigos y limpieza

Corregimos los códigos y sacamos cosas que estaban de ejemplo:

Repository.cs

```
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public class Repository : IRepository {
        public List<Movie> GetMovies() {
            return new List<Movie>() {
                new Movie {Title = "Wakanda forever",
                    ReleaseDate = new DateTime(2022, 11, 11),
                    Poster =
                    "https://upload.wikimedia.org/wikipedia/en/thumb/3/3b/Black_Panther_Wakanda_Forever_poster.jpg/220px-Black_Panther_Wakanda_Forever_poster.jpg"},
                new Movie {Title = "Moana",
                    ReleaseDate = new DateTime(2016, 11, 23),
                    Poster =
                    "https://upload.wikimedia.org/wikipedia/en/thumb/2/26/Moana_Teaser_Poster.jpg/220px-Moana_Teaser_Poster.jpg"},
                new Movie {Title = "Inception",
                    ReleaseDate = new DateTime(2010, 7, 16),
                    Poster =
                    "https://upload.wikimedia.org/wikipedia/en/2/2e/Inception_%282010%29_theatrical_poster.jpg" }
            };
        }
    }
}
```

Movie.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public string Title { get; set; } = null!;
        public DateTime ReleaseDate { get; set; }
        public string Poster { get; set; } = null!;

        public string? TrimmedTitle {
            get {
                if(string.IsNullOrWhiteSpace(Title)) {
                    return null;
                }

                if(Title.Length > 60) {
                    return Title.Substring(0, 60) + "...";
                }
                else {
                    return Title;
                }
            }
        }
    }
}
```

MoviesList.razor

```
@inject IJSRuntime js
```

```
<div style="display:flex;flex-wrap:wrap;align-items:center;">
    <GenericList List="Movies">
        <Loading>
            @Loading
        </Loading>
        <NoRecords>
            @NoRecords
        </NoRecords>
        <HasRecords Context="movie">
            <MovieItem Movie="movie" DeleteMovie="DeleteMovie"/>
        </HasRecords>
    </GenericList>
</div>
```

```
@code {
    [Parameter]
```

```
[EditorRequired]
public List<Movie>? Movies { get; set; }

[Parameter]
public RenderFragment Loading { get; set; } = null!;

[Parameter]
public RenderFragment NoRecords { get; set; } = null!;

private async Task DeleteMovie(Movie movie) {
    var confirmed = await js.Confirm($"Do you want to delete the movie '{movie.Title}'?");

    if (confirmed) {
        Console.WriteLine($"Deleting the movie '{movie.Title}'");
        Movies!.Remove(movie);
    }
}
```

MovielItem.razor

```
<div class="me-2 mb-2" style="text-align:center">
    <a>
        
    </a>
    <p style="max-width:225px;height:44px;font-size:15px;font-weight:bold;">@Movie.Title</p>
    <div>
        <a class="btn btn-info">Edit</a>
        <button type="button" class="btn btn-danger"
            @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
    </div>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }
}
```

Counter.razor.cs

```
using BlazorPeliculas.Client.Helpers;
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] IJSRuntime js { get; set; } = null!;
```

```
private int currentCount = 0;

public void IncrementCount() {
    currentCount++;
}
```

Counter.razor

```
@page "/counter"

<PageTitle>Counter (@currentCount)</PageTitle>
<HeadContent>
    <meta name="description" content="Page to view movies (@currentCount)" />
</HeadContent>
<h1>Counter</h1>

<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
```

Program.cs

```
using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Repositories;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

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

void configureServices(IServiceCollection services) {
    services.AddSingleton< IRepository, Repository>(); /* Inyectar un IRepository.
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
    clase Repository. */
}
```

index.razor

```
@page "/"
@inject IRepository repository
```

```
<PageTitle>Blazor Movies</PageTitle>

<div>
    <h3>Movies</h3>
    <div>
        <MoviesList Movies="Movies">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie> ? Movies{ get; set; }

    protected override void OnInitialized()
    {
        Movies = repository.GetMovies();
    }
}
```

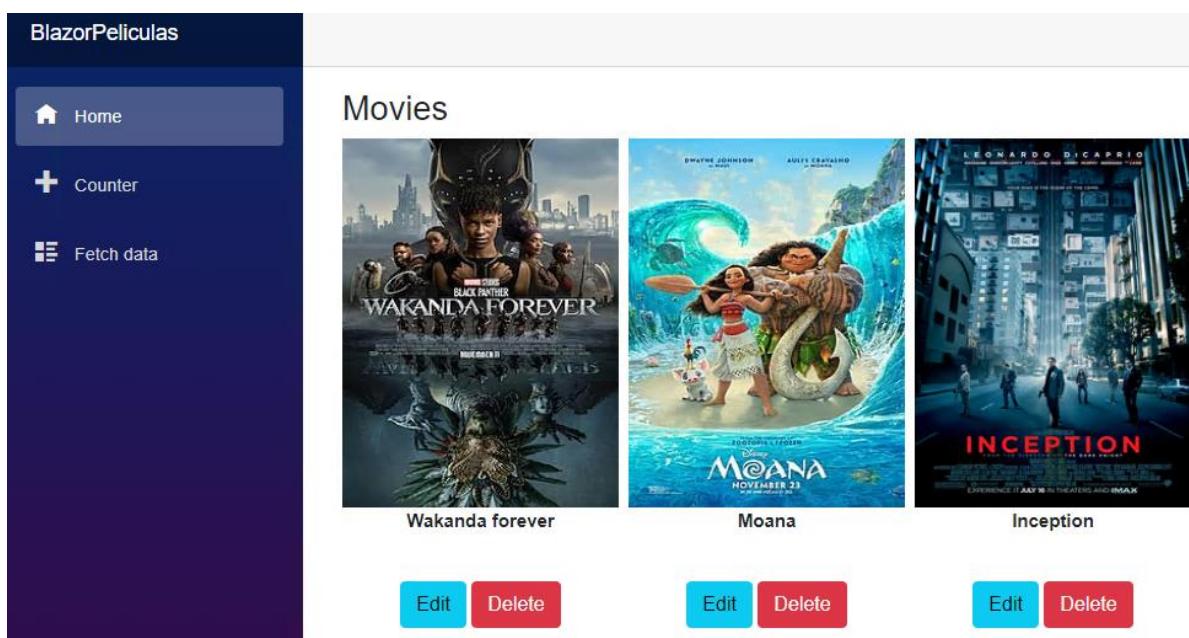
MainLayout.razor

@inherits LayoutComponentBase

```
<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>
```



Page

En una aplicación es normal que queramos que nuestros usuarios puedan navegar entre pantallas. Lo primero que necesitamos para eso es configurar reglas de ruteo en nuestra aplicación. Éstas nos ayudan a definir qué componente va a salir en pantalla según la URL en la que se encuentre el usuario. En Blazor son estas reglas de ruteo se configuran utilizando la directiva `@page`.

Esta directiva funciona de una manera bastante similar al atributo `route` que utilizamos en aplicaciones de ASP.NET Core.

Se puede utilizar más de una directiva `page`.

App.razor

Básicamente esta es una de las piezas fundamentales de una aplicación de Blazor, porque aquí es que se define el enrutador.

```
App.razor
<Router AppAssembly="@typeof(App).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
    <NotFound>
        <PageTitle>Not found</PageTitle>
        <LayoutView Layout="@typeof(MainLayout)">
            <p role="alert">Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>
```

En el código se ve que tenemos el objeto **Router** que utiliza el Assembly de nuestra app con 2 secciones: **Found** y **NotFound**. La primera sección será cuando se encuentre la ruta que estamos queriendo ver a la que le aplicará el layout de **MainLayout**. En **routeData** tendremos datos que podemos mandar a través de la ruta (lo veremos más adelante) por ejemplo, el querystring.

El **FocusOnNavigate** sirve, principalmente para usuarios con problemas en la vista ya que ayuda a sus dispositivos puedan enfocarse en un elemento determinado para que se los pueda leer.

En el **NotFound** se aplica cuando no se encuentra la ruta solicitada. En este caso se podría usar un layout distinto. También podemos definir el **PageTitle** o el mensaje a mostrar.

También podríamos construir un componente que debería mostrarse cuando no se haya encontrado la ruta solicitada. Por ejemplo:

App.razor

```
<Router AppAssembly="@typeof(App).Assembly">
    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
    <NotFound>
        <PageTitle>Not found</PageTitle>
        <LayoutView Layout="@typeof(MainLayout)">
            <CustomNotFound />
        </LayoutView>
    </NotFound>
</Router>
```

CustomNotFound.razor

```
<p role="alert">Sorry, there's nothing at this address.</p>

@code {
    protected override void OnInitialized() {
        Console.WriteLine("404: Route not found");
    }
}
```

La idea de un componente similar sería loguear en una BD cuál fue la ruta no encontrada para análisis posteriores (si hay cientos intentando acceder a una ruta podría indicar que hay algún problema con un link o la ruta en cuestión). Lamentablemente no explicó como acceder a la ruta que falló.

Creamos las carpetas Movies, Genres y Actors en Pages. En ellas, agregamos componentes para crear y editar películas, géneros y actores. En la de Movies,

también agregamos un componente para visualizar. En la MovieItem.razor agregamos el URL para visualizarla.

CustomNotFound.razor

```
<div class="me-2 mb-2" style="text-align:center">
    <a href="@urlMovie">
        
    </a>
    <p style="max-width:225px;height:44px;font-size:15px;font-weight:bold;">
        <a href="@urlMovie" class="text-decoration-none">@Movie.Title</a>
    </p>
    <div>
        <a class="btn btn-info">Edit</a>
        <button type="button" class="btn btn-danger"
            @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
    </div>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }

    private string urlMovie = string.Empty;
    protected override void OnInitialized()
    {
        urlMovie = $"/movie/";
    }
}
```

NavigationManager

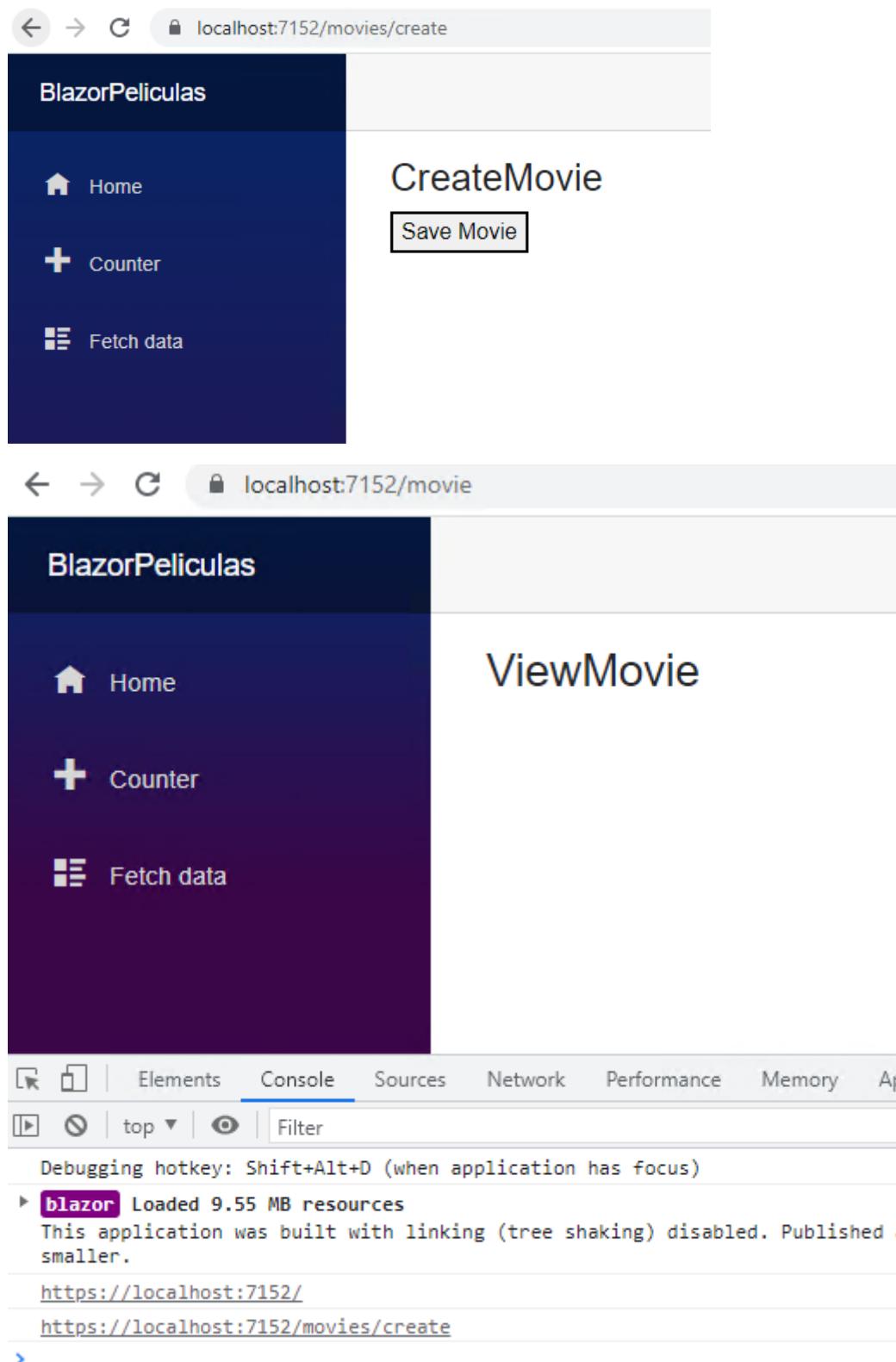
Es un servicio configurado por defecto en nuestra app. Simularemos la creación de una película y luego la navegación a la película creada:

CreateMovie.razor

```
@page "/movies/create"
@inject NavigationManager navManager
<h3>CreateMovie</h3>

<button @onclick="Create">Save Movie</button>

@code {
    void Create() {
        Console.WriteLine(navManager.BaseUri);
        Console.WriteLine(navManager.Uri);
        navManager.NavigateTo("/movie");
    }
}
```



The screenshot displays two Blazor application pages side-by-side, both sharing a common sidebar menu.

Top Page: CreateMovie

- Page Title: CreateMovie
- Content: A "Save Movie" button.
- Sidebar Menu:
 - Home
 - Counter
 - Fetch data

Bottom Page: ViewMovie

- Page Title: ViewMovie
- Content: None (empty page)
- Sidebar Menu:
 - Home
 - Counter
 - Fetch data

Developer Tools Console:

- Logs:
 - Debugging hotkey: Shift+Alt+D (when application has focus)
 - blazor Loaded 9.55 MB resources
 - This application was built with linking (tree shaking) disabled. Published files are smaller.
 - <https://localhost:7152/>
 - <https://localhost:7152/movies/create>

Claramente, no queremos ir a /movie solamente sino a la película en cuestión. Por ejemplo, **movie/1**. Sin embargo, antes debería configurarse el ruteo para poder recibir los parámetros de ruta.

Parámetros de ruta

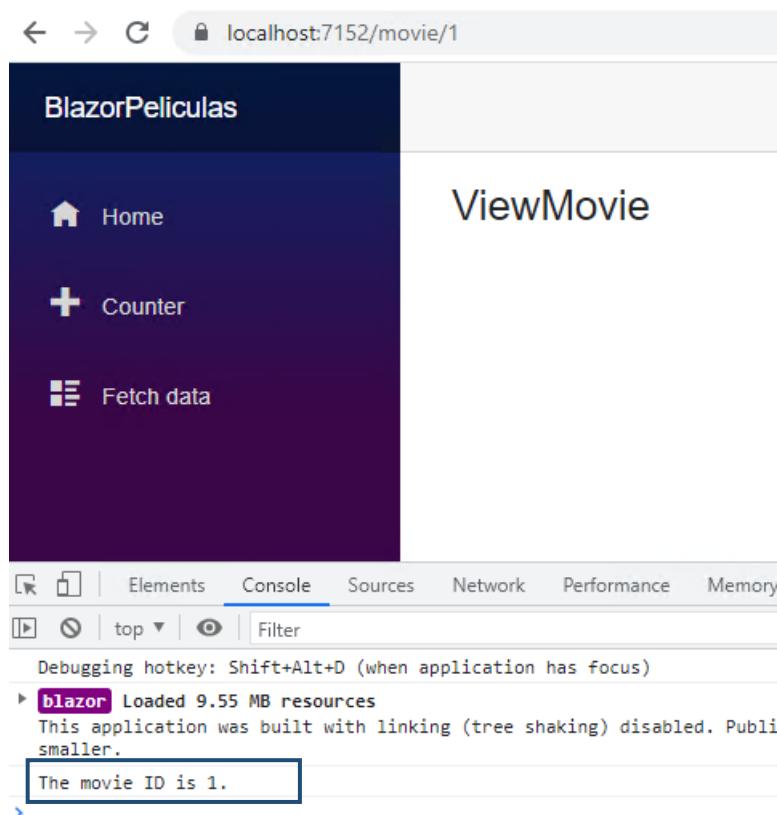
En la directiva de page del componente podemos definir el parámetros que vamos a recibir. A su vez, podemos agregar una restricción indicando que el mismo debe ser entero.

ViewMovie.razor

```
@page "/movie/{MovieID:int}"
<h3>ViewMovie</h3>

@code {
    [Parameter] public int MovieID { get; set; }

    protected override void OnInitialized() {
        Console.WriteLine($"The movie ID is {MovieID}.");
    }
}
```



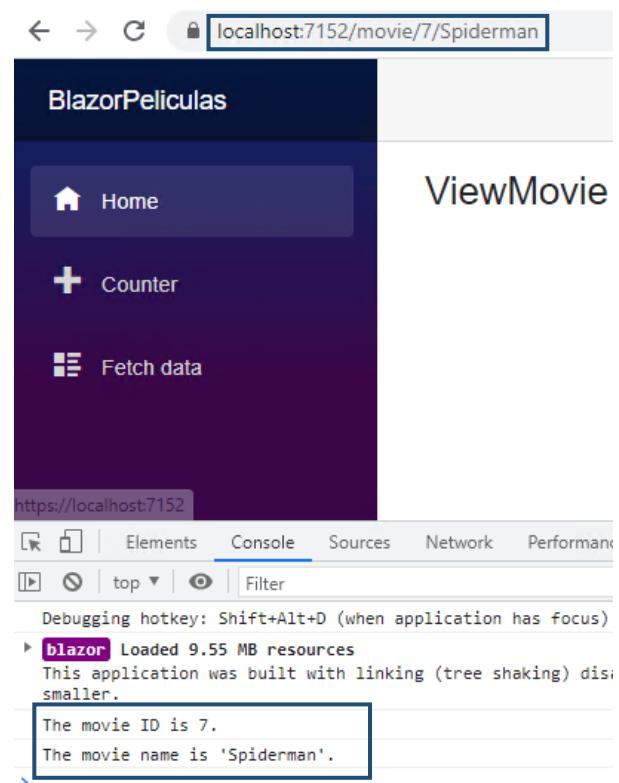
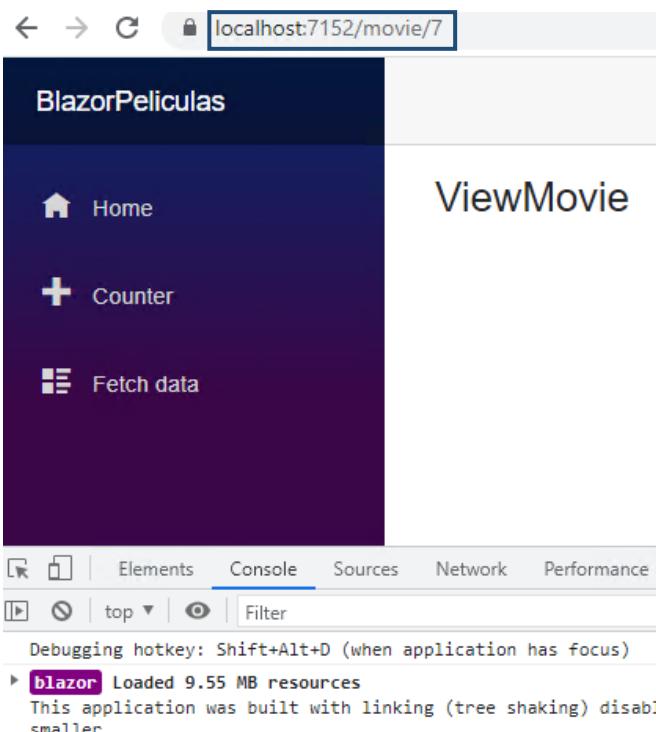
Sin embargo, el ID no le dice nada al usuario. Sería lindo que en el URL se pudiera leer el nombre de la película y recibir el mismo en el código. Para ello:

ViewMovie.razor

```
@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
<h3>ViewMovie</h3>

@code {
    [Parameter] public int MovieID { get; set; }
    [Parameter] public string MovieName { get; set; } = null;

    protected override void OnInitialized()
    {
        Console.WriteLine($"The movie ID is {MovieID}.");
        if(!string.IsNullOrEmpty(MovieName))
        {
            if(MovieName.Length > 1)
                Console.WriteLine($"The movie name is '{MovieName}'.");
        }
    }
}
```



En las imágenes de arriba se pueden ver que se puede acceder a la info de la imagen tanto con el ID sólo como con el ID y el nombre. Agregamos, entonces, los parámetros de ID en las páginas de editar género, actores y películas.

NavLink

Es un componente que nos permite agregar una clase CSS a un elemento link que nos permita marcar cuál es la página en la que estamos navegando.

NavMenu.razor

```
<div class="top-row ps-3 navbar navbar-dark">
<div class="container-fluid">
    <a class="navbar-brand" href="">BlazorPeliculas</a>
    <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
        <span class="navbar-toggler-icon"></span>
    </button>
</div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
<nav class="flex-column">
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
            <span class="oi oi-home" aria-hidden="true"></span> Home
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="counter">
            <span class="oi oi-plus" aria-hidden="true"></span> Counter
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="fetchdata">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="genres">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="actors">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/create">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
        </NavLink>
    </div>
</nav>
</div>
```

```
@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}
```

Vemos que todos los links tienen el componente NavLink al cual le indicamos una clase nav-link. Este componente hará que cuando el href definido en el componente (counter, para el segundo) fuera el mismo que el URL en el que estamos se agregue una clase active al mismo.

En NavMenu agregamos los 3 links para acceder al listado de géneros, actores y crear película. Para los primeros 2, se creó los componentes necesarios.

Lazy loading

Esto nos permite postergar la carga de DLLs. Recordamos que estamos en un contexto de Blazor WebAssembly en el cual tenemos que descargar las DLLs de nuestro proyecto de Client para que podamos utilizar la aplicación en el navegador del usuario.

Sin embargo, no siempre vamos a tener que descargar todas las DLLs al momento de entrar a la aplicación, porque hay librerías que se van a utilizar en momentos particulares.

Entonces, para que la carga de tu aplicación sea más rápida, sería inteligente postergar la carga de esas DLLs que no son necesarias de inmediato para el futuro, para cuando sean necesarias. Para eso, necesitamos instalar un paquete de NuGet. Desde el proyecto de Client hacemos click-derecho y elegimos **Manage NuGet Packages....**

En el tab de Browse buscamos **MathNet.Numerics** que es un paquete de matemáticas y elegimos el que se llama así (aparecen varios parecidos).

```
Counter.razor
@page "/counter"

using BlazorPeliculas.Client.Helpers;
using MathNet.Numerics.Statistics;
using Microsoft.AspNetCore.Components;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
```

```
[Inject] IJSRuntime js { get; set; } = null;

private int currentCount = 0;

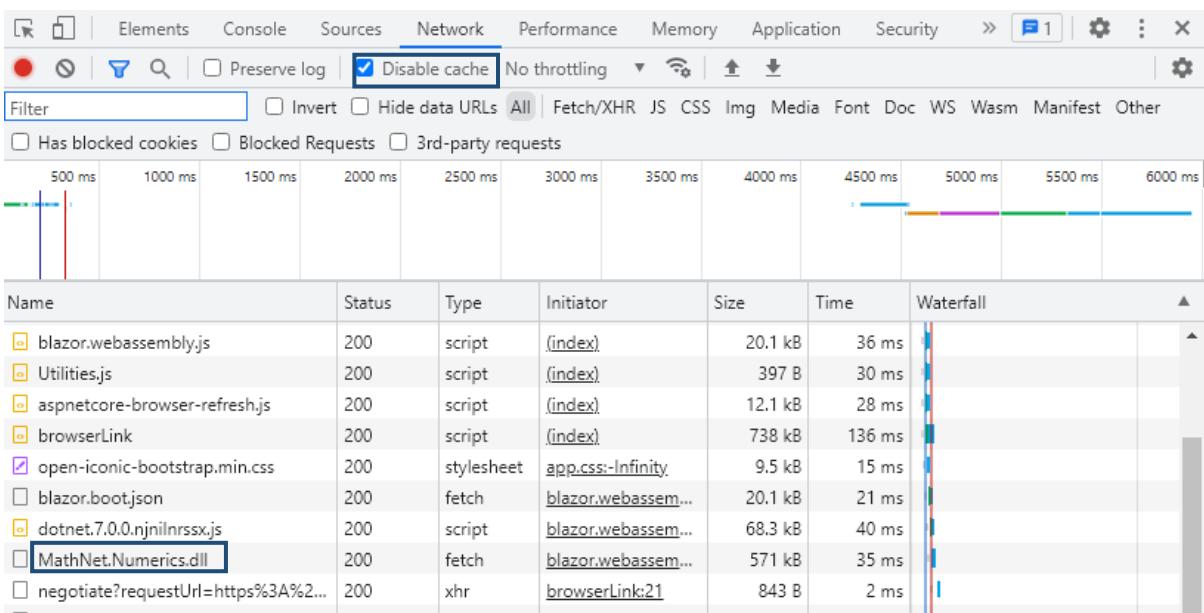
public async Task IncrementCount() {
    var arr = new double[]{ 1, 2, 3, 4, 5};
    var max = arr.Maximum();
    var min = arr.Minimum();

    await js.InvokeVoidAsync("alert", $"The max value is {max} and the min es {min}");
    currentCount++;
}
}
```

The screenshot shows a Blazor application titled "BlazorPelículas". On the left, there's a navigation menu with "Home", "Counter", "Fetch data", and "Genres". The main content area displays two movie posters: "WAKANDA FOREVER" and "MOANA". Below the content, the browser's developer tools are open, specifically the Network tab. A context menu is visible over a row in the Application table, which lists a single entry: "/_framework/MathNetNumerics.dll.sha256...".

#	Name	Respons...	Content...	Content...	Time Ca...	Vary He...
0	/_framework/MathNetNumerics.dll.sha256...	default	application	570240	2023/02/22 10:10:20	

Al ejecutar, limpiamos el cache y luego vamos a Network y deshabilitamos el cache para asegurar que el ejemplo funcione correctamente.



Al recargar la página de Home, vemos que se bajó la DLL de matemática. Idealmente, queríamos que ésta sólo sea descargada si fuéramos a Counter y recién cuándo presionáramos el botón. Al menos, cuando vaya a Counter. Eso lo podemos hacer con **Lazy Loading**. Para ello, lo primero que yo tengo que hacer es bloquear la descarga automática de esa DLL. Entonces, hacemos doble-click sobre el proyecto Client para abrir el csproj y agregar un nuevo ItemGroup:

```

BlazorPeliculas.csproj
<Project Sdk="Microsoft.NET.Sdk.BlazorWebAssembly">

<PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
</PropertyGroup>

<ItemGroup>
    <PackageReference Include="MathNet.Numerics" Version="5.0.0" />
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly" Version="7.0.0" />
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.DevServer"
Version="7.0.0" PrivateAssets="all" />
</ItemGroup>

<ItemGroup>
    <ProjectReference Include=".\\Shared\\BlazorPeliculas.Shared.csproj" />
</ItemGroup>

<ItemGroup>
    <BlazorWebAssemblyLazyLoad Include="MathNet.Numerics.dll" />
</ItemGroup>
</Project>

```

Si refrescáramos la página counter veremos que, ahora, tira un error. ¿Por qué? Porque hemos bloqueado su descarga automática. Para indicar que debe bajar la DLL vamos a App.razor.

Agregamos **OnNavigateAsync** para que llame al método homónimo para que se ejecute cada vez que el usuario navega de una página a otra y **AdditionalAssemblies** para tener el listado de librerías extras disponible.

App.razor

```
@using Microsoft.AspNetCore.Components.WebAssembly.Services;
@using System.Reflection;
@inject LazyAssemblyLoader lazyLoader

<Router AppAssembly="@typeof(App).Assembly"
    OnNavigateAsync="OnNavigateAsync"
    AdditionalAssemblies="assemblies">

    <Found Context="routeData">
        <RouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
    <NotFound>
        <PageTitle>Not found</PageTitle>
        <LayoutView Layout="@typeof(MainLayout)">
            <p role="alert">Sorry, there's nothing at this address.</p>
        </LayoutView>
    </NotFound>
</Router>

@code {
    private List<Assembly> assemblies = new List<Assembly>();
    private async Task OnNavigateAsync(NavigationContext args) {
        if(args.Path.EndsWith("counter")) {
            var loadedAssemblies = await lazyLoader.LoadAssembliesAsync(
                new List<string> { "MathNet.Numerics.dll" }
            );
            assemblies.AddRange(loadedAssemblies);
        }
    }
}
```

Si volvemos a limpiar el cache y asegurarnos que el cache esté deshabilitado. Al dirigirnos a Home vemos que aún no se decargó MathNet. Y recién al dirigirnos a la página de counter, se realiza el download correspondiente.

Resumen

-  Con la directiva `@page` podemos configurar nuestros componentes como ruteables
-  En el archivo `App.Razor` colocamos la configuración del Router de Blazor
-  Con `NavigationManager` podemos redireccionar a los usuarios de nuestra aplicación a distintos lugares
-  Los parámetros de ruta nos sirven para leer desde un componente los valores definidos en la URL
-  El `NavLink` es un componente útil para definir links de menú
-  Lazy loading para retrasar la descarga de DLLs

Formularios

EditForm

Usaremos un componente de Blazor llamado **EditForm** que es muy sencillo cuando está basado en un modelo. Blazor tiene muchos componentes como InputText, InputCheckbox, InputDate, etc que nos ayuda a trabajar directamente con ese tipo de datos.

Definimos, entonces, el modelo Genre.cs en la carpeta Entities del proyecto Shared.

Genre.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Genre {
        public int ID { get; set; }
        public string Name { get; set; } = null!;

    }
}
```

OnSubmit

Se ejecuta cuando el formulario es posteado. Cuando hablamos de postear el formulario hablamos por ejemplo, de cuando el usuario presiona Enter para enviar el formulario o cuando el usuario presiona un botón para enviar los datos del formulario hacia nuestro servidor. Lo que podemos hacer es ejecutar una función a un submit que se va a ejecutar cuando el usuario presione ese botón Enviar.

OnValidSubmit

Para cuando el usuario postea un formulario válido, es decir, que no has roto ninguna de nuestras reglas de validación.

OnInvalidSubmit

Se ejecuta cuando el usuario ha intentado postear un formulario en el cual no se respetan todas las reglas de validación.

CreateGenre.razor

```

@page "/genres/create"
<h3>Create Genre</h3>

<EditForm Model="genre" OnSubmit="Create">
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-bind-Value="@genre.Name" />
        </div>
    </div>

    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private void Create() {
        Console.WriteLine("Executing Create method");
        Console.WriteLine($"Genre name: {genre.Name}");
    }
}

```

GenresList.razor

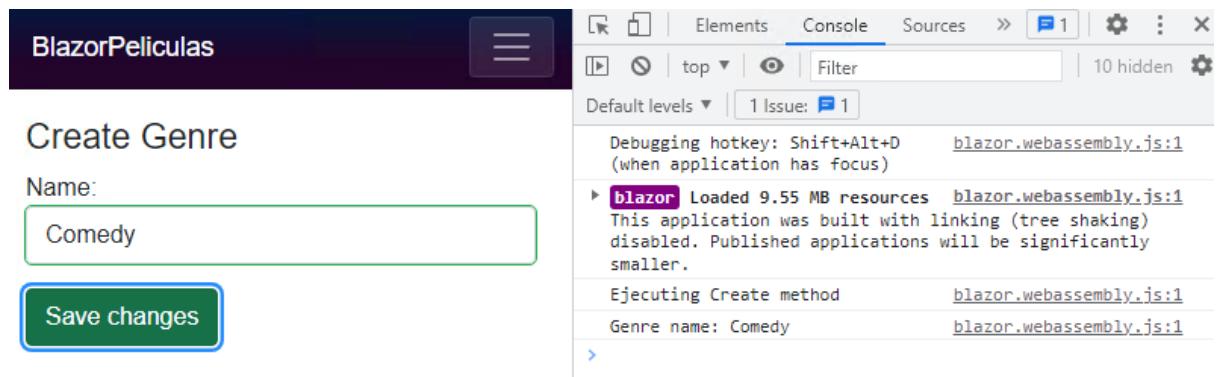
```

@page "/genres"
<h3>Genres</h3>

<div class="mb-3">
    <a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

@code {
}

```



Sin embargo, este código permite ejecutar el método del botón si el textbox estuviera vacío. Así, se estaría creando un género sin nombre.

Validaciones

Las reglas de validación nos permiten definir las reglas que nuestra aplicación va a verificar que se cumplan. Éstas pueden ser definidas a nivel de un modelo o a nivel de la base de datos e incluso a nivel de nuestra aplicación en la capa de negocios.

Por ahora vamos a concentrarnos en validaciones a nivel del modelo. Podemos utilizar validaciones por atributo para definir reglas de validación relacionadas con nuestra entidad.

Por ejemplo digamos que queremos que el campo **Name** sea requerido.

Genre.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Genre {
        public int ID { get; set; }

        [Required]
        public string Name { get; set; } = null!;
    }
}
```

CreateGenre.razor

```
@page "/genres/create"
<h3>Create Genre</h3>

<EditForm Model="genre" OnValidSubmit="Create">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-bind-Value="@genre.Name" />
        </div>
    </div>

    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
```

```
@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private void Create() {
        Console.WriteLine("Ejecuting Create method");
        Console.WriteLine($"Genre name: {genre.Name}");
    }
}
```

Vale aclarar que es necesario cambiar el **OnSubmit** por **OnValidSubmit** porque la idea es que el método sólo se ejecute si se cumplieron todas las reglas. **DataAnnotationsValidator** es un componente que se va a encargar de forzar las validaciones que provengan de anotaciones de datos (por ejemplo, el Required del campo Name de la clase Genre)

Si intentamos crear un género vacío, ahora, no se leerá nada en la consola (porque el método sólo se ejecuta si todas las reglas fueron cumplidas) y el campo **Name** cambiará su borde a rojo. Sin embargo, no hay mayores datos de que es lo que está mal al respecto.

Para evitar esto, se agrega el componente **ValidationMessage** con una expresión lambda que provocará que se coloquen todos los errores de validación que estén asociados con el campo **Name**.

CreateGenre.razor

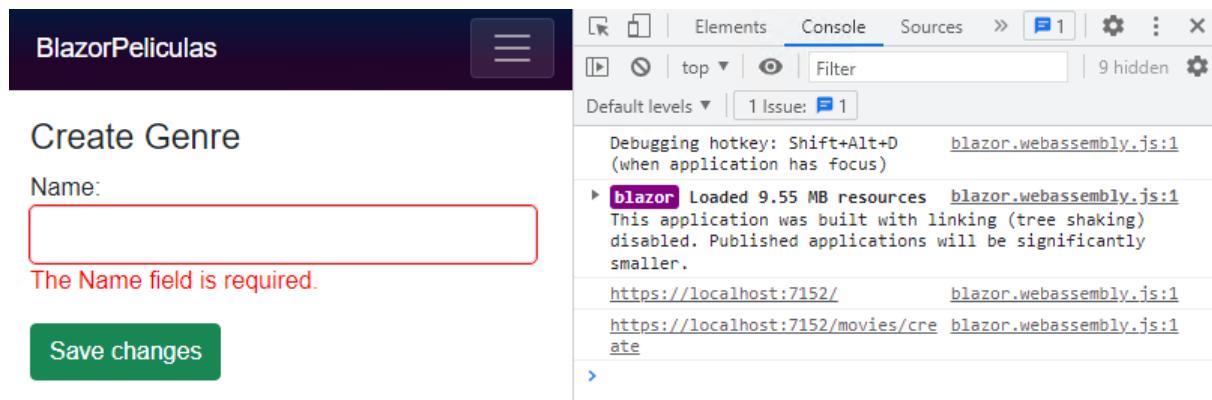
```
@page "/genres/create"
<h3>Create Genre</h3>

<EditForm Model="genre" OnValidSubmit="Create">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-bind-Value="@genre.Name" />
            <ValidationMessage For="@(() => genre.Name)" />
        </div>
    </div>
    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
```

```
private void Create() {
    Console.WriteLine("Ejecuting Create method");
    Console.WriteLine($"Genre name: {genre.Name}");
}
```



The screenshot shows a Blazor application titled "BlazorPelículas". A modal dialog is open with the heading "Create Genre". In the "Name:" input field, there is no value, and a red border surrounds it with the error message "The Name field is required." below it. A green "Save changes" button is visible at the bottom of the modal. To the right of the modal, the browser's developer tools are open, specifically the "Console" tab. It shows a single issue: "blazor.Loaded 9.55 MB resources blazor.webassembly.js:1". Below this, there is a detailed message about tree shaking and published applications being smaller. At the bottom of the console, two URLs are listed: "https://localhost:7152/" and "https://localhost:7152/movies/cre ate".

Si queremos personalizar el mensaje de error:

```
Genre.cs
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPelículas.Shared.Entities {
    public class Genre {
        public int ID { get; set; }

        [Required(ErrorMessage = "The field '{0}' is required.")]
        public string Name { get; set; } = null!;
    }
}
```

The screenshot shows a Blazor application titled "BlazorPelículas". On the left, there's a form titled "Create Genre" with a text input field labeled "Name". The input field is empty and has a red border, indicating it's required. Below the input field is the error message "The field 'Name' is required.". To the right of the form is a browser's developer tools console tab. It displays several messages from the "blazor" logger. One message says "Loaded 9.55 MB resources" and another says "Ejecuting Create method". There is also a message about tree shaking being disabled.

Si escribiéramos un nombre, al salir del textbox, desaparecería el mensaje de error y el borde cambiaría a verde. Al clickear en el botón, se ejecuta el método correspondiente.

The screenshot shows the same Blazor application after a name has been entered into the "Name" field. The field now has a green border, indicating it's valid. The error message "The field 'Name' is required." is no longer visible. In the browser console, the "Ejecuting Create method" message is present, along with a "Genre name: Comedia" message, indicating the name has been successfully submitted.

Al pensar el componente de edición, detectamos que el formulario es igual. Es un textbox en el que se debe ingresar el nombre y debe tener las mismas validaciones. Para evitar repeticiones, crearemos el componente **GenreForm.razor**.

```
GenreForm.razor
<EditForm Model="Genre" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Genre.Name" />
            <ValidationMessage For="@(() => Genre.Name)" />
        </div>
    </div>

    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>

@code {
    [Parameter]
```

```
[EditorRequired]
public Genre Genre { get; set; } = null!;

[Parameter]
[EditorRequired]
public EventCallback OnValidSubmit { get; set; }
}
```

CreateGenre.razor

```
@page "/genres/create"
<h3>Create Genre</h3>
<GenreForm Genre="genre" OnValidSubmit="Create" />

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private void Create() {
        Console.WriteLine("Ejecuting Create method");
        Console.WriteLine($"Genre name: {genre.Name}");
    }
}
```

EditGenre.razor

```
@page "/genres/edit/{GenreID:int}"
<h3>EditGenre</h3>

<GenreForm Genre="Genre" OnValidSubmit="Edit" />

@code {
    [Parameter] public int GenreID { get; set; }
    private Genre? Genre;

    protected override void OnInitialized() {
        //Eventualmente vendrá de una BD, por ahora... va hardcodeado
        Genre = new Genre() {
            ID = GenreID,
            Name = "Comedy"
        };
    }

    private void Edit() {
        Console.WriteLine("Ejecuting Edit method");
        Console.WriteLine($"Genre ID: {Genre!.ID}");
        Console.WriteLine($"Genre Name: {Genre!.Name}");
    }
}
```

Con estos cambios, si volvemos a probar vemos que el alta sigue funcionando. Al dirigirnos a un URL de edición de género:

Vemos que en la consola se imprime el ID que se recuperó, que se muestra inicialmente el nombre Comedy (hardcodeado) y que en consola se imprime el valor que recuperó el formulario del OnValidSubmit.

¿Salvar los cambios?

Si queremos evitar que se pierdan los cambios no salvados al intentar navegar desde una ruta a otra (dentro de la misma app) usamos el componente **NavigationLock** y **OnBeforeInternalNavigation**.

Necesitamos también, cambiar la forma de manejo del formulario. Debemos cambiar el modelo por un **EditContext**. Este componente, además de tener una copia del modelo tiene el estado del mismo. De esta forma, podremos saber si el formulario ha cambiado. Si el estado de nuestro **editContext** dice que no ha habido cambios, no hay necesidad de consultar nada al intentar navegar a otra página interna.

Otra de las cosas a tener en cuenta es que luego de crear un género nuevo, querríamos navegar hacia el listado. Por lo tanto, es necesario tener una variable que diga si fue posteado correctamente para que no me pregunte si quiero guardar los cambios antes de permitir re-dirigirme.

```
GenreForm.razor
```

```
@inject IJSRuntime js
```

```
<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation"></NavigationLock>
<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
```

```
<div>
    <InputText class="form-control" @bind-Value="@Genre.Name" />
    <ValidationMessage For="@(() => Genre.Name)" />
</div>
</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>

@code {
    private EditContext editContext = null!;

    protected override void OnInitialized() {
        editContext = new(Genre);
    }

    [Parameter]
    [EditorRequired]
    public Genre Genre { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    public bool formPostedCorrectly { get; set; } = false;

    private async Task OnBeforeInternalNavigation(LocationChangingContext context) {
        var formWasEdited = editContext.IsModified();

        if(!formWasEdited)
            return;

        if(formPostedCorrectly)
            return;

        var confirmed = await js.Confirm("Do you want to abandon the page and lose the changes?");
        if(confirmed)
            return;

        context.PreventNavigation();
    }
}
```

En el CreateGenre, tenemos que hacer el inject del **NavigationManager** para permitirnos dirigirnos a la página índice luego de grabarse correctamente. A su vez, necesitamos agregar el **@ref** para setear la referencia al componente y poder cambiar su variable **formPostedCorrectly** con true.

CreateGenre.razor

```
@page "/genres/create"
@inject NavigationManager navManager

<h3>Create Genre</h3>

<GenreForm @ref="genreForm" Genre="genre" OnValidSubmit="Create" />

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío
    private GenreForm? genreForm;

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private void Create() {
        Console.WriteLine("Ejecuting Create method");
        Console.WriteLine($"Genre name: {genre.Name}");

        genreForm!.formPostedCorrectly = true;
        navManager.NavigateTo("/genres");
    }
}
```

Hacemos similar con el EditGenre.

EditGenre.razor

```
@page "/genres/edit/{GenreID:int}"
@inject NavigationManager navManager
<h3>EditGenre</h3>

<GenreForm @ref="genreForm" Genre="Genre" OnValidSubmit="Edit" />

@code {
    [Parameter] public int GenreID { get; set; }
    private Genre? genre;
    private GenreForm? genreForm;

    protected override void OnInitialized() {
        //Eventualmente vendrá de una BD, por ahora... va hardcodeado
        Genre = new Genre() {
            ID = GenreID,
            Name = "Comedy"
        };
    }

    private void Edit() {
        Console.WriteLine("Ejecuting Edit method");
        Console.WriteLine($"Genre ID: {Genre!.ID}");
        Console.WriteLine($"Genre Name: {Genre!.Name}");
    }
}
```

```
genreForm!.formPostedCorrectly = true;
navManager.NavigateTo("/genres");
}
}
```

SweetAlert2

Lo mejor de Blazor es que tenemos lo mejor de los 2 mundos. Lo mejor de .NET y lo mejor de Javascript. Veremos SweetAlert2 que es una herramienta de JS.

Vamos a <https://sweetalert2.github.io/>

Si bien es de JS, hay un paquete de NuGet que lo encapsula. Buscamos el paquete razor/sweetalert2 (de CurrieTechnologies) y lo instalamos. Luego hacemos unos cambios para utilizarlo.

Program.cs

```
using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Repositories;
using CurrieTechnologies.Razor.Sweetalert2;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

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

void configureServices(IServiceCollection services) {
    services.AddSingleton< IRepository, Repository>(); /* Inyectar un IRepository.
        Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
        clase Repository. */
    services.AddSweetAlert2();
}
```

Vamos a <https://github.com/Basaingeal/Razor.Sweetalert2> para copiar el código de Add the script tag.

Index.html

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
```

```
<title>BlazorPeliculas</title>
<base href="/" />
<link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
<link href="css/app.css" rel="stylesheet" />
<link rel="icon" type="image/png" href="favicon.png" />
<link href="BlazorPeliculas.Client.styles.css" rel="stylesheet" />
</head>

<body>
<div id="app">
<svg class="loading-progress">
<circle r="40%" cx="50%" cy="50%" />
<circle r="40%" cx="50%" cy="50%" />
</svg>
<div class="loading-progress-text"></div>
</div>

<div id="blazor-error-ui">
    An unhandled error has occurred.
    <a href="" class="reload">Reload</a>
    <a class="dismiss">X </a>
</div>
<script src="_framework/blazor.webassembly.js"></script>
<script src="js/Utilities.js"></script>
<script src="_content/CurrieTechnologies.Razor.Sweetalert2/sweetalert2.min.js"></script>
</body>

</html>
```

Y el **using** en el **_Imports.razor**.

```
_Imports.razor
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorPeliculas.Client
@using BlazorPeliculas.Client.Shared
@using BlazorPeliculas.Client.Utilities
@using BlazorPeliculas.Shared.Entities
@using BlazorPeliculas.Client.Repositories
@using BlazorPeliculas.Client.Helpers
@using CurrieTechnologies.Razor.Sweetalert2
```

Y ya podemos usarlo en nuestro formulario

GenreForm.razor

```
@inject SweetAlertService swal

<NavigationLock OnBeforeInternalNavigation="OnBeforeInternalNavigation"></NavigationLock>
<EditForm EditContext="editContext" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Genre.Name" />
            <ValidationMessage For="@(() => Genre.Name)" />
        </div>
    </div>

    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>

@code {
    private EditContext editContext = null!;

    protected override void OnInitialized() {
        editContext = new(Genre);
    }
    [Parameter]
    [EditorRequired]
    public Genre Genre { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    public bool formPostedCorrectly { get; set; } = false;

    private async Task OnBeforeInternalNavigation(LocationChangingContext context) {
        var formWasEdited = editContext.IsModified();

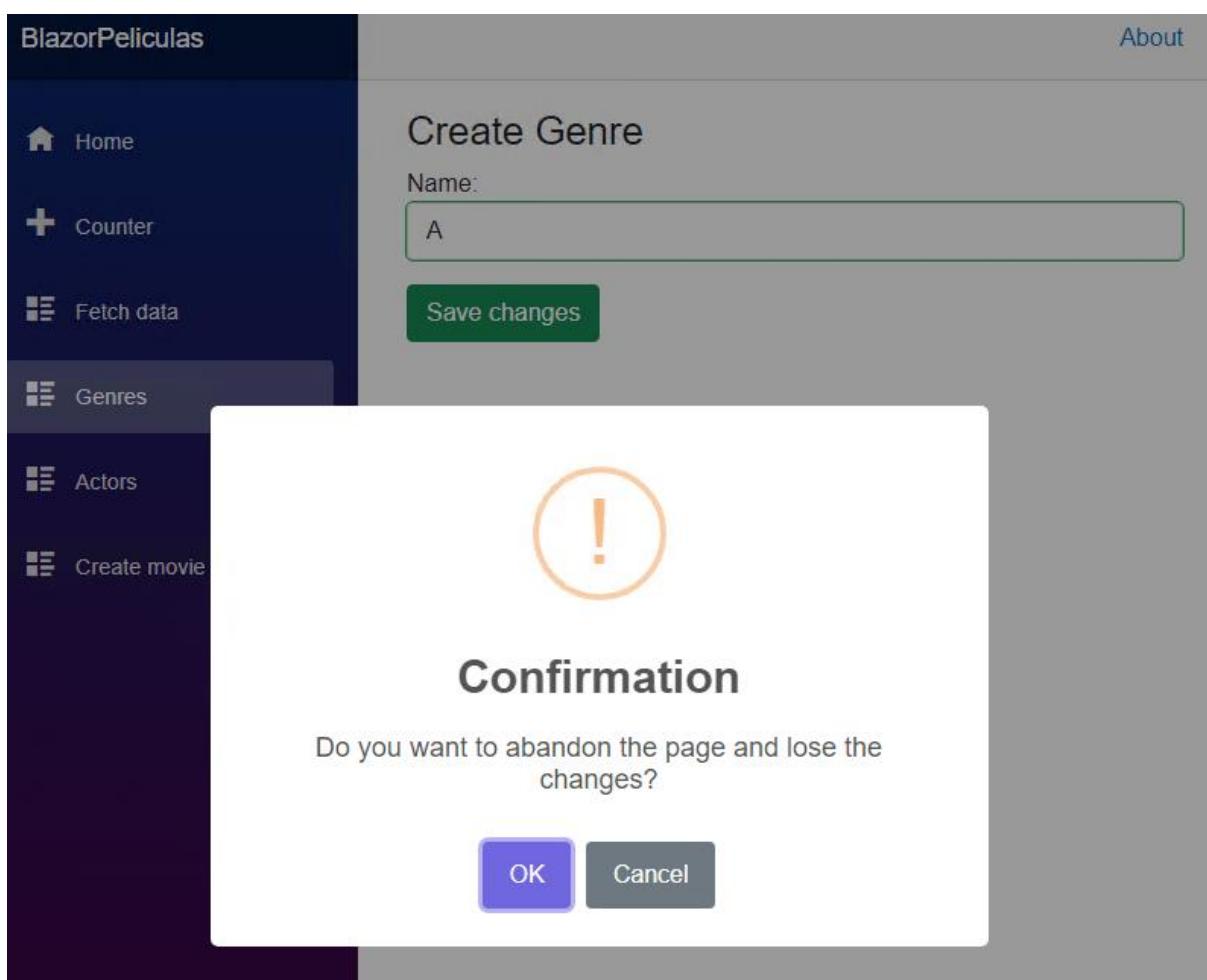
        if(!formWasEdited)
            return;

        if(formPostedCorrectly)
            return;

        var result = await swal.FireAsync(new SweetAlertOptions
        {
            Title = "Confirmation",
            Text = "Do you want to abandon the page and lose the changes?",
            Icon = SweetAlertIcon.Warning,
            ShowCancelButton = true
        })
    }
}
```

```
});  
  
var confirmed = !string.IsNullOrEmpty(result.Value);  
if(confirmed)  
    return;  
  
context.PreventNavigation();  
}  
}
```

Al escribir algo en el textbox y tratar de cambiar a otra página recibimos un mensaje más bonito:



Filtro de películas

Creamos un componente nuevo al que le seteamos la directiva `page` correspondiente para que sea ruteable. A su vez injectamos el repositorio ya que necesitamos el listado de películas.

NavMenu.razor

```
<div class="top-row ps-3 navbar navbar-dark">
<div class="container-fluid">
    <a class="navbar-brand" href="">BlazorPeliculas</a>
    <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
        <span class="navbar-toggler-icon"></span>
    </button>
</div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
<nav class="flex-column">
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
            <span class="oi oi-home" aria-hidden="true"></span> Home
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="counter">
            <span class="oi oi-plus" aria-hidden="true"></span> Counter
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="fetchdata">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
        </NavLink>
    </div>

    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/filter">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Movies filter
        </NavLink>
    </div>

    <div class="nav-item px-3">
        <NavLink class="nav-link" href="genres">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="actors">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/create">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
        </NavLink>
    </div>
</nav>
```

```
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}
```

FilterMovie.razor

```
@page "/movies/filter"
@inject IRepository repository
<h3>Movies filter</h3>

<div class="row g-3 align-items-center mb-3">
    <div class="col-sm-3">
        <input type="text" class="form-control" id="title" placeholder="Movie title"
            @bind-value="Title" @bind-value:event="oninput"
            @onkeypress="@(KeyboardEventArgs e) => TitleKeyPress(e)" />
    </div>
    <div class="col-sm-3">
        <select class="form-select" @bind="Genre">
            <option value="0">-- Select a genre --</option>
            @foreach(var item in genres) {
                <option value="@item.ID">@item.Name</option>
            }
        </select>
    </div>
    <div class="col-sm-6" style="display:flex;">
        <div class="form-check me-2">
            <input type="checkbox" class="form-check-input" id="premieres" @bind="futurePremieres" />
            <label class="form-check-label" for="premieres">Future premieres</label>
        </div>
        <div class="form-check me-2">
            <input type="checkbox" class="form-check-input" id="billboard" @bind="onBillboard" />
            <label class="form-check-label" for="billboard">On billboard</label>
        </div>
        <div class="form-check">
            <input type="checkbox" class="form-check-input" id="mostVoted" @bind="mostVoted" />
            <label class="form-check-label" for="mostVoted">Most voted</label>
        </div>
    </div>
    <div class="col-12">
        <button type="button" class="btn btn-primary" @onclick="FilteredMovies">Filter</button>
    </div>

```

```
<button type="button" class="btn btn-danger" @onclick="Clean">Clean</button>
</div>
</div>

<MoviesList Movies="Movies"></MoviesList>

@code {
    private string Title = "";
    private string Genre = "0";
    private List<Genre> genres = new List<Genre>();
    private bool futurePremieres = false;
    private bool onBillboard = false;
    private bool mostVoted = false;
    private List<Movie>? Movies;

    protected override void OnInitialized() {
        Movies = repository.GetMovies();
    }

    private void TitleKeyPress(KeyboardEventArgs e) {
        if(e.Key == "Enter") {
            FilteredMovies();
        }
    }

    private void FilteredMovies() {
        Movies = repository.GetMovies()
            .Where(x => x.Title.ToLower().Contains	Title.ToLower()).ToList();

        Console.WriteLine($"Title: {Title}");
        Console.WriteLine($"Genre: {Genre}");
        Console.WriteLine($"onBillboard: {onBillboard}");
        Console.WriteLine($"futurePremieres: {futurePremieres}");
        Console.WriteLine($"mostVoted: {mostVoted}");
    }

    private void Clean() {
        Movies = repository.GetMovies();
        Title = "";
        Genre = "0";
        futurePremieres = false;
        onBillboard = false;
        mostVoted = false;
    }
}
```

Formulario de actores

Insertar imagen

El componente **InputFile** acepta un parámetro **multiple** que permite seleccionar más de 1 archivo. Para mostrarle la imagen al usuario usaremos un formato llamado BASE64.

Con el método **StateHasChanged** forzamos al componente a que muestre los cambios en la interfaz de usuario.

InputImg.razor

```
<div>
    <label>@Label</label>
    <div>
        <InputFile OnChange="OnChange" accept=".jpg,.jpeg,.png" />
    </div>
</div>

<div>
    @if(imageBase64 is not null) {
        <div>
            <div style="margin:10px;">
                
            </div>
        </div>
    }
    @if (ImageURL is not null) {
        <div>
            <div style="margin:10px;">
                
            </div>
        </div>
    }
</div>

@code {
    [Parameter] public string Label { get; set; } = "Image";
    [Parameter] public string? ImageURL { get; set; }
    [Parameter] public EventCallback<string> SelectedImage { get; set; }
    private string? imageBase64;

    async Task OnChange(InputFileChangedEventArgs e) {
        var images = e.GetMultipleFiles();

        foreach(var image in images) {
            var arrBytes = new byte[image.Size];
            await image.OpenReadStream().ReadAsync(arrBytes);
            imageBase64 = Convert.ToString(arrBytes);
            ImageURL = null; //Si editando el actor cambiamos la imagen, borramos el URL anterior
                            //para que desaparezca la foto anterior.
        }
    }
}
```

```
    await SelectedImage.InvokeAsync(imageBase64);
    StateHasChanged();
}
}
```

Vale aclarar que mandar una imagen al servidor es pesado. Sólo debe hacerse cuando sea necesario. Si el usuario está en modo edición y quiere editar un actor existente que ya tiene una foto pero **no la cambia**, pues no hay que mandarla al servidor porque estaríamos mandando lo que ya existe en el servidor.

Por tanto, para ser eficientes, lo que hay que hacer es nulificar (cargarla en una variable "local" y limpiar el dato del componente (null)).

ActorsList.razor

```
@page "/actors"
<h3>Actors list</h3>

<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>

@code {
```

Actors.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Actor {
        public int ID { get; set; }
        [Required]
        public string Name { get; set; } = null!;
        public string? Bio { get; set; }
        public string? Photo { get; set; }
        public DateTime? BirthDate { get; set; }

    }
}
```

CreateActor.razor

```
@page "/actors/create"
<h3>Create Actor</h3>

<ActorsForm OnValidSubmit="Create" Actor="Actor" />
@code {
    private Actor Actor = new Actor();

    void Create() {
        Console.WriteLine("Create actor");
    }
}
```

ActorsForm.razor

```
<EditForm Model="Actor" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Actor.Name" />
            <ValidationMessage For="@(() => Actor.Name)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Birthdate:</label>
        <div>
            <InputDate class="form-control" @bind-Value="@Actor.BirthDate" />
            <ValidationMessage For="@(() => Actor.BirthDate)" />
        </div>
    </div>

    <div class="mb-3">
        <InputImg Label="Photo" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
    </div>
    <div class="mb-3">
    </div>

    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    string? ImageURL;

    protected override void OnInitialized() {
        if (!string.IsNullOrEmpty(Actor.Photo)) {
            ImageURL = Actor.Photo;
            Actor.Photo = null;          //Al editar, seteamos el URL y limpiamos el dato.
            //Si no se le carga uno nuevo, no será re-enviado.
    }
}
```

```
        }
    }
    private void SelectedImage (string imageBase64) {
        Actor.Photo = imageBase64;
        ImageURL = null;
    }
    //private EditContext editContext = null;

    [Parameter]
    [EditorRequired]
    public Actor Actor { get; set; } = null;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }
}
```

EditActor.razor

```
@page "/actors/edit/{ActorID:int}"
<h3>Edit Actor</h3>

<ActorsForm Actor="Actor" OnValidSubmit="Edit" />

@code {
    [Parameter] public int ActorID { get; set; }
    Actor Actor = new Actor();

    protected override void OnInitialized() {
        Actor = new Actor()
        {
            ID = ActorID,
            Name = "Emiliano",
            BirthDate = DateTime.Today
        };
    }

    void Edit() {
        Console.WriteLine("Editing the actor");
    }
}
```

Componente de Markdown

Para la carga de la biografía de los actores crearemos un componente que nos permita cargar datos con formato. Para ello usaremos Markdown (MD).

Creamos el componente **InputMD** en Shared. Como los componentes son clases, podemos heredar de ellos. Así, entonces, heredamos de **InputTextarea** para poder

tener toda su funcionalidad y agregarle cosas. **CurrentValue**, por ejemplo, es heredado del **InputTextArea**. Usaremos **typeparam** para hacerlo genérico y poder validar cualquier campo.

Para poder mostrar el preview crearemos el componente **ShowMD** en Shared e instalamos el NuGet package llamado **Markdig** (Markdig Mutel).

ShowMD.razor

```
@using Markdig

@if(string.IsNullOrEmpty(MDContent)) {
    if (LoadingTemplate is not null) {
        @LoadingTemplate
    }
}
else {
    if(!string.IsNullOrEmpty(HTMLContent)) {
        @((MarkupString) HTMLContent)
    }
}

@code {
    [Parameter] public string? MDContent { get; set; }
    [Parameter] public RenderFragment LoadingTemplate { get; set; } = null!;
    private string HTMLContent = null!;

    protected override void OnParametersSet() {
        if (MDContent is not null)
            HTMLContent = Markdown.ToHtml(MDContent);
    }
}
```

InputMD.razor

```
@using System.Linq.Expressions;
@inherits InputTextArea
@typeparam TValue

<div>
    <label>@Label</label>
    <div>
        <InputTextArea @bind-Value="CurrentValue" />
        <ValidationMessage For="For" />
    </div>
</div>

<div>
    <label>@Label (preview)</label>
    <div class="markdown-container" style="overflow:auto;">
        <ShowMD MDContent="@CurrentValue" >
    </div>
</div>
```

```
</ShowMD>
</div>
</div>
@code {
    [Parameter] public Expression<Func< TValue >>? For { get; set; }
    [Parameter] public string Label { get; set; } = "Field";
}
```

InputMD.razor

```
@using System.Linq.Expressions;
@inherits InputTextArea
@typeparam TValue

<div>
    <label>@Label</label>
    <div>
        <InputTextArea @bind-Value="CurrentValue" />
        <ValidationMessage For="For" />
    </div>
</div>

<div>
    <label>@Label (preview)</label>
    <div class="markdown-container" style="overflow:auto;">
        <ShowMD MDContent="@CurrentValue">
            </ShowMD>
        </div>
    </div>
@code {
    [Parameter] public Expression<Func< TValue >>? For { get; set; }
    [Parameter] public string Label { get; set; } = "Field";
}
```

ActorsForm.razor

```
<EditForm Model="Actor" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Name:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Actor.Name" />
            <ValidationMessage For="@(() => Actor.Name)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Birthdate:</label>
```

```

<div>
    <InputDate class="form-control" @bind-Value="@Actor.BirthDate" />
    <ValidationMessage For="@(() => Actor.BirthDate)" />
</div>
</div>

<div class="mb-3">
    <InputImg Label="Photo" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
</div>
<div class="mb-3 form-markdown">
    <InputMD @bind-Value="@Actor.Bio"
        For=@(() => Actor.Bio)
        Label="Bio" />
</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    string? ImageURL;

    protected override void OnInitialized() {
        if (!string.IsNullOrEmpty(Actor.Photo)) {
            ImageURL = Actor.Photo;
            Actor.Photo = null;      //Al editar, seteamos el URL y limpiamos el dato.
                                    //Si no se le carga uno nuevo, no será re-enviado.
        }
    }
    private void SelectedImage (string imageBase64) {
        Actor.Photo = imageBase64;
        ImageURL = null;
    }
    //private EditContext editContext = null!;

    [Parameter]
    [EditorRequired]
    public Actor Actor { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }
}
    
```

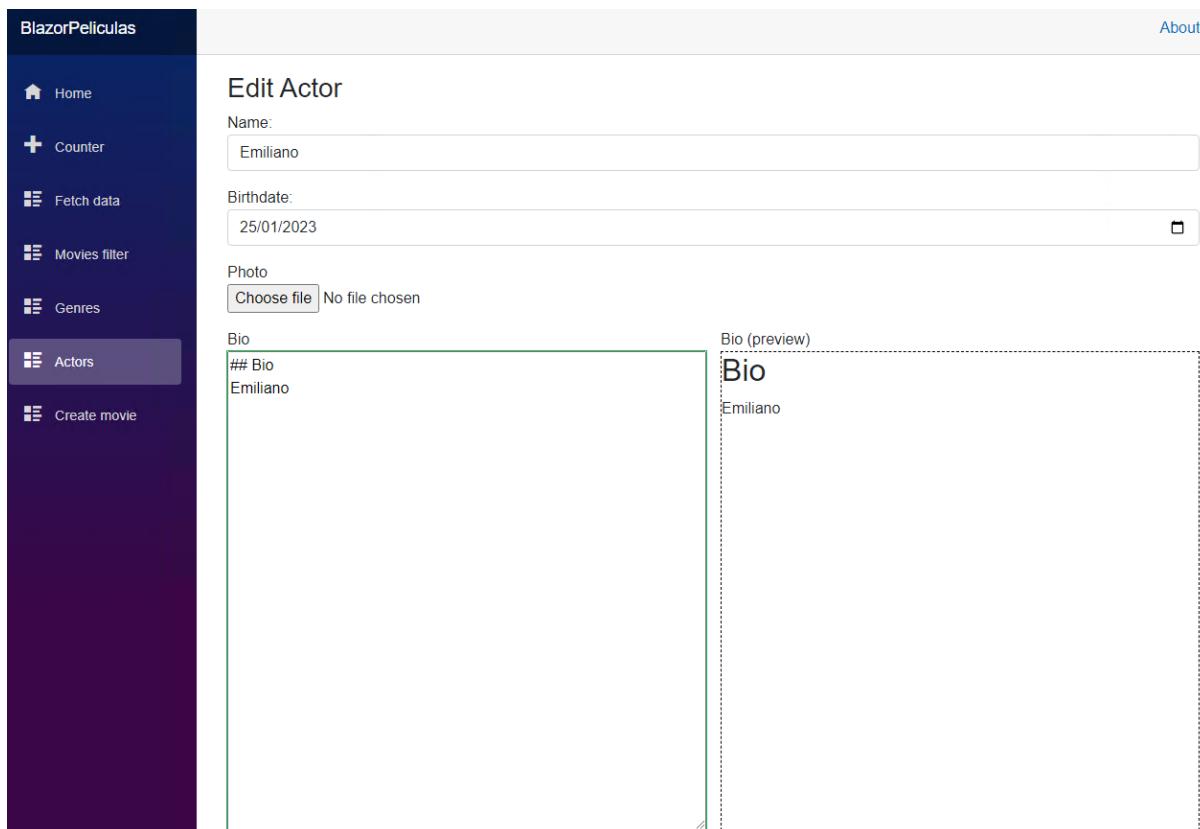
Agregamos al final del [wwwroot/css/app.css](#)

```

app.css
.form-markdown {
    display:flex;
}
    
```

```
.form-markdown textarea {
    width:500px;
    height:500px;
    margin-right:15px;
}

.form-markdown .markdown-container {
    border: 1px dashed black;
    width: 500px;
    height: 500px;
}
```



Formulario de películas

Agregamos algunos parámetros a la clase [Movies.cs](#).

```
Movie.cs
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
```

```

public class Movie {
    public int ID { get; set; }
    [Required]
    public string Title { get; set; } = null!;
    public string? Summary { get; set; }
    public bool OnBillboard { get; set; }
    public string? Trailer { get; set; }
    public DateTime? ReleaseDate { get; set; }
    public string? Poster { get; set; }

    public string? TrimmedTitle {
        get {
            if(string.IsNullOrWhiteSpace(Title)) {
                return null;
            }

            if>Title.Length > 60) {
                return Title.Substring(0, 60) + "...";
            }
            else {
                return Title;
            }
        }
    }
}

```

Creamos el formulario de películas

MoviesForm.razor

```

<EditForm Model="Movie" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />

    <div class="mb-3">
        <label>Title:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Movie.Title" />
            <ValidationMessage For="@(() => Movie.Title)" />
        </div>
    </div>

    <div class="mb-3">
        <label>On billboard:</label>
        <div>
            <InputCheckbox @bind-Value="@Movie.OnBillboard" />
            <ValidationMessage For="@(() => Movie.OnBillboard)" />
        </div>
    </div>

```

```

<div class="mb-3">
    <label>Release date:</label>
    <div>
        <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
        <ValidationMessage For="@(() => Movie.ReleaseDate)" />
    </div>
</div>

<div class="mb-3">
    <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
</div>

<div class="mb-3 form-markdown">
    <InputMD @bind-Value="@Movie.Summary"
        For="@(() => Movie.Summary)"
        Label="Summary" />
</div>

    <button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    string? ImageURL;

    protected override void OnInitialized() {
        if(!string.IsNullOrEmpty(Movie.Poster)) {
            ImageURL = Movie.Poster;
            Movie.Poster = null;      //Al editar, seteamos el URL y limpiamos el dato.
            //Si no se le carga uno nuevo, no será re-enviado.
        }
    }
    private void SelectedImage(string imageBase64) {
        Movie.Poster = imageBase64;
        ImageURL = null;
    }
}

```

Y cambiamos los componentes existentes para que lo utilicen:

CreateMovie.razor

```

@page "/movies/create"
@inject NavigationManager navManager
<h3>Create Movie</h3>

```

```
<MoviesForm Movie="Movie" OnValidSubmit="Create" />
@code {
    private Movie Movie = new Movie();

    void Create() {
        Console.WriteLine(navManager.BaseUri);
        Console.WriteLine(navManager.Uri);
        navManager.NavigateTo("/");
    }
}
```

EditMovie.razor

```
@page "/movies/edit/{MovieID:int}"
<h3>Edit Movie</h3>

<MoviesForm Movie="Movie" OnValidSubmit="Edit" />
@code {
    [Parameter] public int MovieID { get; set; }
    private Movie Movie = new Movie();

    protected override void OnInitialized() {
        Movie = new Movie()
        {
            ID = MovieID,
            Title = "Title"
        };
    }

    private void Edit() {
        Console.WriteLine("Editing movie...");
    }
}
```

Selección múltiple

Para conseguir esto, necesitaremos una clase en Helpers llamada [MultipleSelectorModel.cs](#).

MultipleSelectorModel.cs

```
namespace BlazorPeliculas.Client.Helpers {
    public class MultipleSelectorModel {
        public MultipleSelectorModel(string key, string value) {
            Key = key;
            Value = value;
        }
        public string Key { get; set; }
        public string Value { get; set; }
    }
}
```

}

Con esto podemos crear el componente **MultipleSelector.razor** (en **/Client/Shared**)

MultipleSelector.razor

```
<div class="multiple-selector">
    <ul class="selectable-ul">
        @foreach(var item in Unselected) {
            <li @onclick=@(() => Select(item))>@item.Value</li>
        }
    </ul>
    <div class="multiple-selector-buttons">
        <button type="button" @onclick="SelectAll">></button>
        <button type="button" @onclick="UnselectAll">&lt;&lt;</button>
    </div>
    <ul class="selectable-ul">
        @foreach (var item in Selected) {
            <li @onclick=@(() => Unselect(item))>@item.Value</li>
        }
    </ul>
</div>

@code {
    [Parameter] public List<MultipleSelectorModel> Unselected { get; set; } =
        new List<MultipleSelectorModel>();
    [Parameter] public List<MultipleSelectorModel> Selected { get; set; } =
        new List<MultipleSelectorModel>();

    private void Select(MultipleSelectorModel item) {
        Unselected.Remove(item);
        Selected.Add(item);
    }

    private void Unselect(MultipleSelectorModel item) {
        Selected.Remove(item);
        Unselected.Add(item);
    }

    private void SelectAll() {
        Selected.AddRange(Unselected);
        Unselected.Clear();
    }

    private void UnselectAll() {
        Unselected.AddRange(Selected);
        Selected.Clear();
    }
}
```

También agregamos un **GenresMovie.cs** (en **/Shared/Entities**) para definir nuestra entidad de pares Película/Género. Una película puede tener más de un género por lo que tendremos un listado de esta asociación.

GenresMovie.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class GenresMovie {
        public int MovieID { get; set; }
        public int GenreID { get; set; }
    }
}
```

Ahora, agregamos este listado en nuestra entidad de película (**/Shared/Entities/Movies.cs**)

Movie.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public int ID { get; set; }
        [Required]
        public string Title { get; set; } = null!;
        public string? Summary { get; set; }
        public bool OnBillboard { get; set; }
        public string? Trailer { get; set; }
        public DateTime? ReleaseDate { get; set; }
        public string? Poster { get; set; }
        public List<GenresMovie> GenresMovie { get; set; } = new List<GenresMovie>();
        public string? TrimmedTitle {
            get {
                if(string.IsNullOrWhiteSpace(Title)) {
                    return null;
                }

                if>Title.Length > 60 {
                    return Title.Substring(0, 60) + "...";
                }
                else {
                    return Title;
                }
            }
        }
    }
}
```

```
        }
    }
}
}
```

Agregamos al final del [wwwroot/css/app.css](#)

```
app.css
.multiple-selector {
    display:flex;
}

.selectable-ul {
    height:200px;
    overflow-y:auto;
    list-style-type:none;
    width:170px;
    padding:0;
    border-radius:3px;
    border:1px solid #ccc;
}

.selectable-ul li {
    cursor:pointer;
    border-bottom:1px #eee solid;
    padding:2px 10px;
    font-size:14px;
}

.selectable-ul li:hover {
    background-color:#08c;
}

.multiple-selector-buttons {
    display:flex;
    flex-direction:column;
    justify-content:center;
    padding:5px;
}

.multiple-selector-buttons button {
    margin:5px;
}
```

Al editar una película, ya tendremos géneros seleccionados y no seleccionados. Al editar, es necesario los parámetros **Selected** y **Unselected** con sus listados. Es necesario aclarar que la película tendrá un listado de géneros asociados, pero el

componente **MultipleSelectorModel** recibe un listado de modelos. Por lo tanto, es necesario traer el listado de géneros y crear un listado de modelos para poder pasárselo al componente.

MoviesForm.razor

```
<EditForm Model="Movie" OnValidSubmit="OnValidSubmit">
<DataAnnotationsValidator />

<div class="mb-3">
    <label>Title:</label>
    <div>
        <InputText class="form-control" @bind-Value="@Movie.Title" />
        <ValidationMessage For="@(() => Movie.Title)" />
    </div>
</div>

<div class="mb-3">
    <label>On billboard:</label>
    <div>
        <InputCheckbox @bind-Value="@Movie.OnBillboard" />
        <ValidationMessage For="@(() => Movie.OnBillboard)" />
    </div>
</div>

<div class="mb-3">
    <label>Release date:</label>
    <div>
        <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
        <ValidationMessage For="@(() => Movie.ReleaseDate)" />
    </div>
</div>

<div class="mb-3">
    <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
</div>

<div class="mb-3 form-markdown">
    <InputMD @bind-Value="@Movie.Summary"
        For="@(() => Movie.Summary)"
        Label="Summary" />
</div>

<div class="mb-3">
    <label>Genres:</label>
    <div>
        <MultipleSelector Selected="Selected" Unselected="Unselected" />
    </div>
</div>

<button class="btn btn-success" type="submit">Save changes</button>
```

```

</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    [Parameter]
    public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
    [Parameter]
    [EditorRequired]
    public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();

    private List<MultipleSelectorModel> Selected { get; set; } = new List<MultipleSelectorModel>();
    private List<MultipleSelectorModel> Unselected { get; set; } = new List<MultipleSelectorModel>();

    string? ImageURL;

    protected override void OnInitialized() {
        if(!string.IsNullOrEmpty(Movie.Poster)) {
            ImageURL = Movie.Poster;
            Movie.Poster = null;      //Al editar, seteamos el URL y limpiamos el dato.
                                    //Si no se le carga uno nuevo, no será re-enviado.
        }

        Selected = SelectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(), x.Name)).ToList();
        Unselected = UnselectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(),
x.Name)).ToList();
    }

    private void SelectedImage(string imageBase64) {
        Movie.Poster = imageBase64;
        ImageURL = null;
    }
}

```

Agregamos el uso del nuevo componente en CreateMovie:

```

CreateMovie.razor
@page "/movies/create"
@inject NavigationManager navManager
<h3>Create Movie</h3>

<MoviesForm Movie="Movie" OnValidSubmit="Create"
    UnselectedGenres="Unselected"/>
@code {
    private Movie Movie = new Movie();
    private List<Genre> Unselected = new List<Genre>();
}

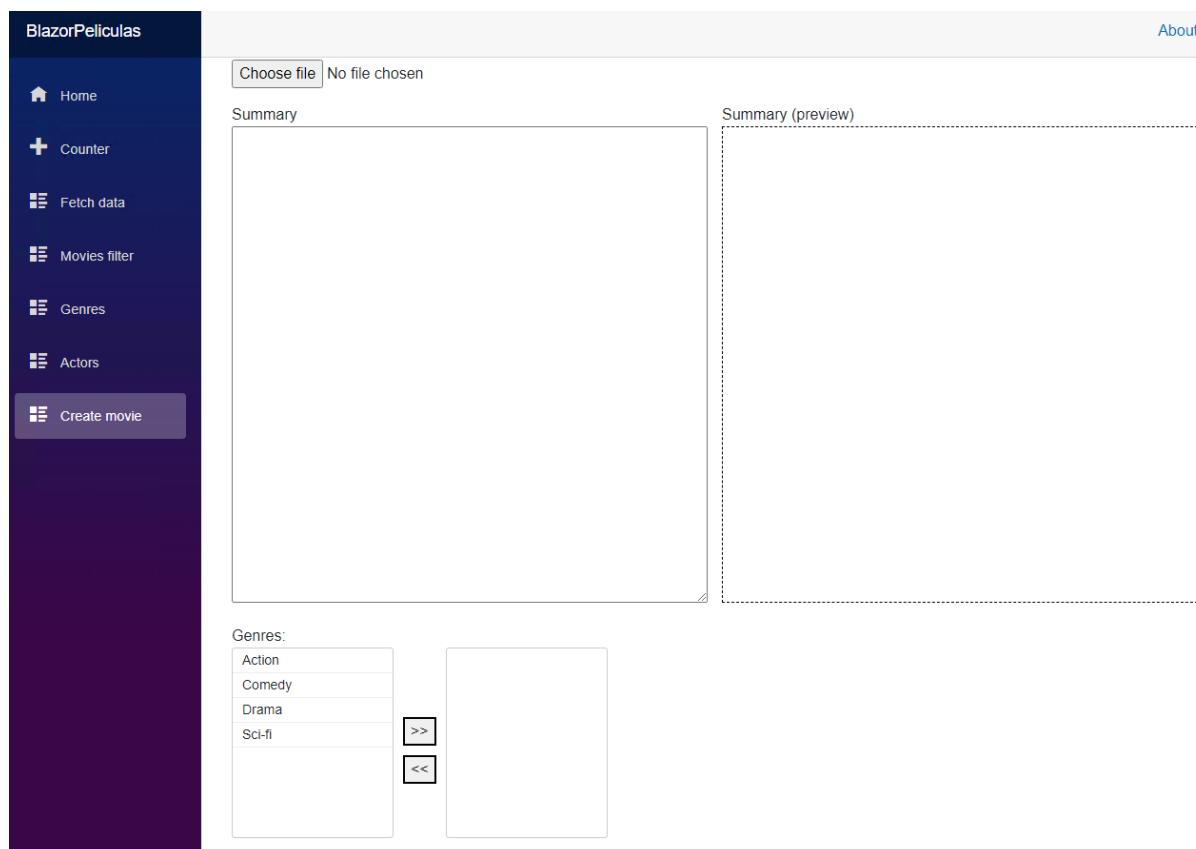
```

```

protected override void OnInitialized() {
    //Por el momento está hardcodeado...
    Unselected = new List<Genre>() {
        new Genre() { ID = 1,Name = "Comedy" },
        new Genre() { ID = 2,Name = "Drama" },
        new Genre() { ID = 3,Name = "Action" },
        new Genre() { ID = 4,Name = "Sci-fi" }
    };
}

void Create() {
    Console.WriteLine(navManager.BaseUri);
    Console.WriteLine(navManager.Uri);
    navManager.NavigateTo("/");
}
}

```



Y lo mismo con con [EditMovie.razor](#).

EditMovie.razor

```

@page "/movies/edit/{MovieID:int}"
<h3>Edit Movie</h3>

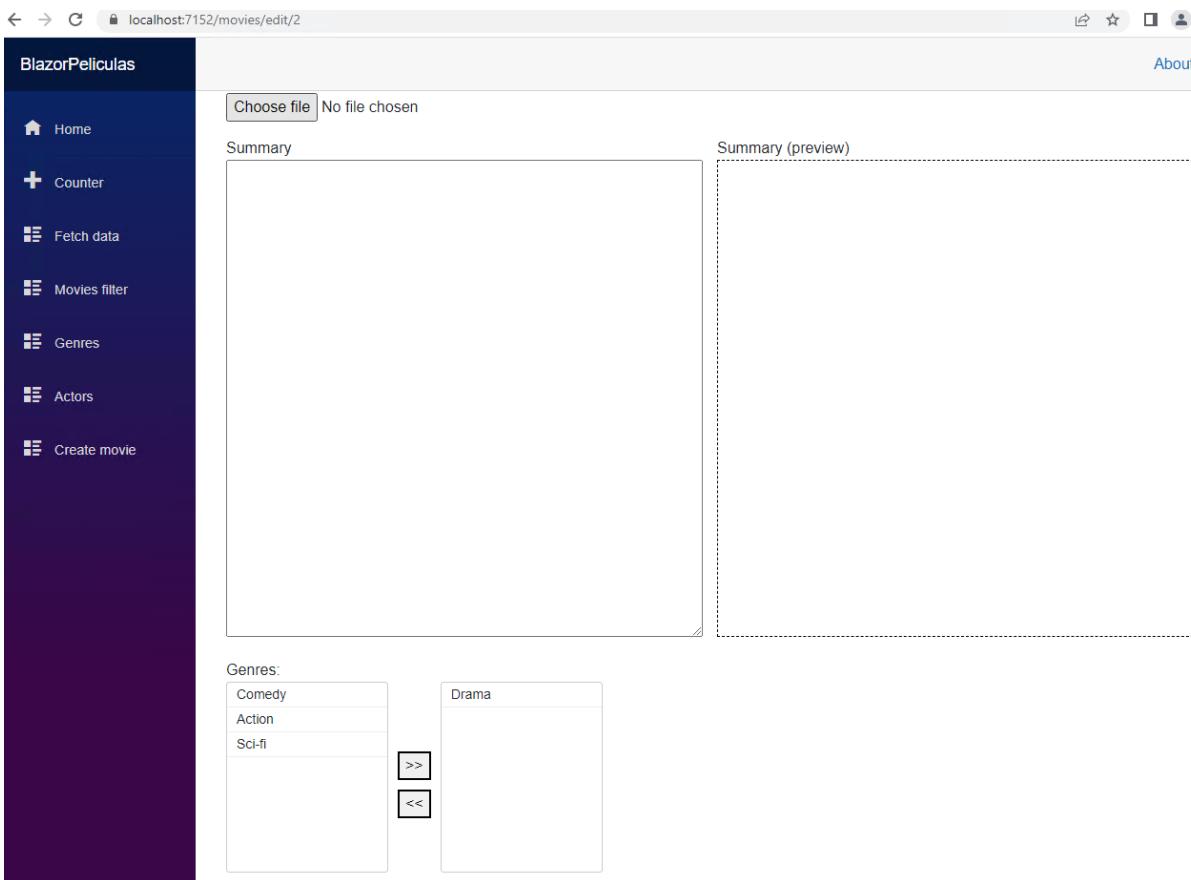
```

```
<MoviesForm Movie="Movie" OnValidSubmit="Edit"
    UnselectedGenres="Unselected"
    SelectedGenres="Selected" />

@code {
    [Parameter] public int MovieID { get; set; }
    private Movie Movie = new Movie();
    private List<Genre> Unselected = new List<Genre>();
    private List<Genre> Selected = new List<Genre>();

    protected override void OnInitialized() {
        Movie = new Movie()
        {
            ID = MovieID,
            Title = "Title"
        };
        //Por el momento está hardcodeado...
        Unselected = new List<Genre>()
        {
            new Genre() { ID = 1, Name = "Comedy" },
            new Genre() { ID = 3, Name = "Action" },
            new Genre() { ID = 4, Name = "Sci-fi" }
        };
        Selected = new List<Genre>()
        {
            new Genre() { ID = 2, Name = "Drama" }
        };
    }

    private void Edit() {
        Console.WriteLine("Editing movie...");
    }
}
```



En la imagen anterior vemos que Drama está como género seleccionado tal como está hardcodeado en el código.

Componente de selección múltiple

Para la selección de actores utilizaremos un componente de selección múltiple. **Typeahead**, por ejemplo, es básicamente cuando tú tienes un textbox, tú escribes en él y según lo que tú escribas te salen sugerencias. Vamos a <https://github.com/Blazored/Typeahead> para instalarlo. En ella dice que se puede instalar bajando el paquete NuGet **Blazored.Typeahead**. En esa página dice que se deben agregar las siguientes líneas en el **index.html**.

```
<link href="_content/Blazored.Typeahead/blazored-typeahead.css"
rel="stylesheet" />
```

```
<script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>
```

```
index.html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, user-
```

```

scalable=no" />
<title>BlazorPeliculas</title>
<base href="/" />
<link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
<link href="css/app.css" rel="stylesheet" />
<link rel="icon" type="image/png" href="favicon.png" />
<link href="BlazorPeliculas.Client.styles.css" rel="stylesheet" />
<link href="_content/Blazored.Typeahead/blazored-typeahead.css" rel="stylesheet" />
</head>

<body>
<div id="app">
    <svg class="loading-progress">
        <circle r="40%" cx="50%" cy="50%" />
        <circle r="40%" cx="50%" cy="50%" />
    </svg>
    <div class="loading-progress-text"></div>
</div>

<div id="blazor-error-ui">
    An unhandled error has occurred.
    <a href="" class="reload">Reload</a>
    <a class="dismiss">X</a>
</div>
<script src="_framework/blazor.webassembly.js"></script>
<script src="js/Utilities.js"></script>
<script src="_content/CurrieTechnologies.Razor.Sweetalert/sweetalert2.min.js"></script>
<script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>
</body>

</html>

```

En este componente tendremos, además de la selección múltiple con typeahead tendremos un listado arrastable. El componente **BlazoredTypeahead** tiene un parámetro **TValue** y **TItem** que nos permiten indicar el dato que se utilizará. En nuestro caso, será **T** para que sea genérico.

Al definir el parámetro **SearchMethod** (que es una función que recibe un string y devuelve un **IEnumerable** del tipo **T**) vemos que los atributos se los puede separar por coma (**Parameter**, **EditorRequired**)

El evento **ValueChange** se dispara cuando el usuario elige una de las opciones mostradas. **ValueExpression** es básicamente una expresión que apunta hacia el valor, en este caso dummy.

En nuestro caso **T** serán actores. Para poder comparar actores, es necesario codificar como se comparan. Tenemos que sobreescibir los métodos **Equals** y **GetHashCode**.

Actor.cs

```

using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Actor {
        public int ID { get; set; }
        [Required]
        public string Name { get; set; } = null!;
        public string? Bio { get; set; }
        public string? Photo { get; set; }
        public DateTime? BirthDate { get; set; }

        public override bool Equals(object? obj) {
            if(obj is Actor a2) {
                //El objeto recibido es un actor
                return ID == a2.ID;
            }
            return false;
        }

        public override int GetHashCode() {
            return base.GetHashCode();
        }
    }
}

```

Typeahead nos obliga agregar **SelectedTemplate** por lo que lo agregaremos aunque no lo utilizaremos. **ResultTemplate** es un RenderFragment porque el componente es genérico. En nuestro caso, en éste podemos indicar que queremos mostrar acerca de los elementos encontrados: la foto y el nombre, por ejemplo. Algo similar ocurre con **ListTemplate**.

MultipSelectorTypeahead.razor

```

@using Blazored.Typeahead
@typeparam T

<BlazoredTypeahead TValue="T" TItem="T" SearchMethod="SearchMethod"
    Value="dummy" ValueChanged="SelectedElement"
    ValueExpression="@(() => dummy)">
    <ResultTemplate>
        @ResultTemplate(context)
    </ResultTemplate>
    <NotFoundTemplate>
        @((MarkupString) NotFoundTemplate)
    </NotFoundTemplate>

```

```

<SelectedTemplate></SelectedTemplate>
</BlazoredTypeahead>

<ul class="list-group">
    @foreach(var item in SelectedElements) {
        <li draggable="true"
            @ondragstart="@(() => HandleDragStart(item))"
            @ondragover="@(() => HandleDragover(item))"
            class="list-group-item list-group-item-action">
            @ListTemplate(item)
            <span
                @onclick="@(() => SelectedElements.Remove(item))"
                class="badge bg-primary rounded-pill" style="cursor:pointer">
                X
            </span>
        </li>
    }
</ul>

@code {
    [Parameter, EditorRequired] public Func<string, Task<IEnumerable<T>>> SearchMethod { get;set;} = null;
    [Parameter] public List<T> SelectedElements { get; set; } = new List<T>();
    [Parameter, EditorRequired] public RenderFragment<T> ResultTemplate { get; set; } = null!;
    [Parameter, EditorRequired] public RenderFragment<T> ListTemplate { get; set; } = null!;
    [Parameter, EditorRequired] public string NotFoundTemplate { get; set; } = "No records were found.";

    T? dummy = default(T);
    T? draggedItem;

    private void SelectedElement(T item) {
        if(!SelectedElements.Any(x => x.Equals(item))) {
            //Es un elemento que no fue seleccionado anteriormente.
            SelectedElements.Add(item);
        }

        //Para que al seleccionar se limpie el textbox
        dummy = default(T);
    }

    private void HandleDragStart(T item) {
        draggedItem = item;
    }

    private void HandleDragover(T item) {
        if(draggedItem is null)
            return;

        if(!item.Equals(draggedItem)) {
            var draggedIndex = SelectedElements.IndexOf(draggedItem);
        }
    }
}

```

```

        var itemIndex = SelectedElements.IndexOf(item);
        SelectedElements[itemIndex] = draggedItem;
        SelectedElements[draggedIndex] = item;
    }
}
}
}

```

Agregaremos el componente creado en el **MoviesForm**.

MoviesForm.razor

```

<EditForm Model="Movie" OnValidSubmit="OnValidSubmit">
    <DataAnnotationsValidator />

    <div class="mb-3">
        <label>Title:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Movie.Title" />
            <ValidationMessage For="@(() => Movie.Title)" />
        </div>
    </div>

    <div class="mb-3">
        <label>On billboard:</label>
        <div>
            <InputCheckbox @bind-Value="@Movie.OnBillboard" />
            <ValidationMessage For="@(() => Movie.OnBillboard)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Release date:</label>
        <div>
            <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
            <ValidationMessage For="@(() => Movie.ReleaseDate)" />
        </div>
    </div>

    <div class="mb-3">
        <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
    </div>

    <div class="mb-3 form-markdown">
        <InputMD @bind-Value="@Movie.Summary"
            For=@(() => Movie.Summary)
            Label="Summary" />
    </div>

    <div class="mb-3">
        <label>Genres:</label>
    </div>

```

```

<div>
    <MultipleSelector Selected="Selected" Unselected="Unselected"></MultipleSelector>
</div>
</div>

<div class="mb-3">
    <label>Actors:</label>
    <div>
        <MultipleSelectorTypeahead Context="Actor" SearchMethod="SearchActors"
            SelectedElements="SelectedActors" NotFoundTemplate="No actors were found">
            <ListTemplate>
                @Actor.Name
            </ListTemplate>
            <ResultTemplate>
                
                @Actor.Name
            </ResultTemplate>
        </MultipleSelectorTypeahead>
    </div>
</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    [Parameter]
    public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
    [Parameter]
    [EditorRequired]
    public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();

    [Parameter]
    public List<Actor> SelectedActors { get; set; } = new List<Actor>();

    private List<MultipleSelectorModel> Selected { get; set; } = new List<MultipleSelectorModel>();
    private List<MultipleSelectorModel> Unselected { get; set; } = new List<MultipleSelectorModel>();

    string? ImageURL;

    protected override void OnInitialized()
    {
        if(!string.IsNullOrEmpty(Movie.Poster))
        {
            ImageURL = Movie.Poster;
            Movie.Poster = null;           //Al editar, seteamos el URL y limpiamos el dato.
    }
}

```

```

        //Si no se le carga uno nuevo, no será re-enviado.
    }

    Selected = SelectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(), x.Name)).ToList();
    Unselected = UnselectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(),
x.Name)).ToList();
}

private void SelectedImage(string imageBase64) {
    Movie.Poster = imageBase64;
    ImageURL = null;
}

private async Task<IEnumerable<Actor>> SearchActors(string searchText) {
    //Por el momento va hardcodeado...
    return new List<Actor>() {
        new Actor {ID=1, Name="Tom Holland", Photo =
"https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Tom_Holland_by_Gage_Skidmore.jpg/1200px-Tom_Holland_by_Gage_Skidmore.jpg"},
        new Actor {ID=2, Name="Tom hanks", Photo =
"https://upload.wikimedia.org/wikipedia/commons/a/a9/Tom_Hanks_TIFF_2019.jpg"}
    };
}
}

```

Genres:

Comedy
Drama
Action
Sci-fi

>>

<<

Actors:

Tom
Tom Holland
Tom hanks

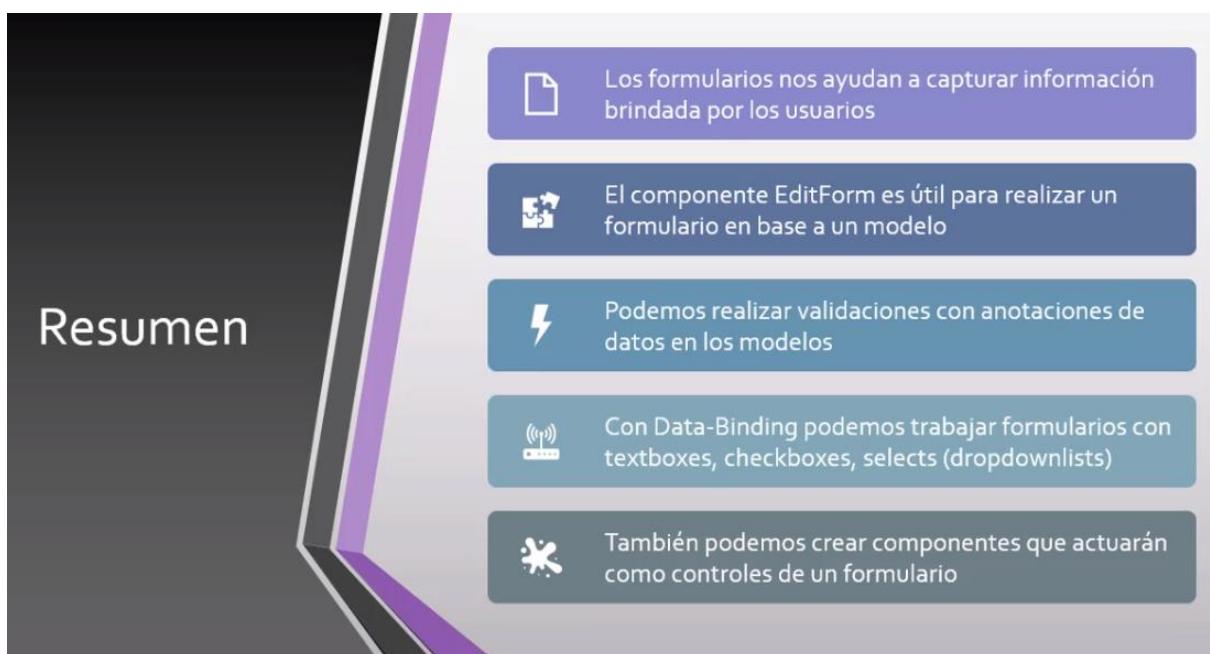
<https://localhost:7152/movies/filter>

En la imagen anterior se ve que se pueden seleccionar los actores y en la siguiente se ve que está su **X** para permitir eliminarlo. A su vez, se pueden cambiar el orden arrastrando un elemento sobre el otro.

Actors:

Tom Holland 
Tom hanks 

Save changes



HttpClient

Normalmente una aplicación web del lado del cliente va a querer comunicarse con un web API o servicio web. Esto porque en el web API podemos colocar lógica de procesamiento y almacenamiento de datos que no queremos que resida en el cliente de nuestra aplicación.

Para comunicarnos con un web API utilizamos el protocolo HTTP. Existe un servicio en Blazor que nos ayuda con esa tarea: **HttpClient**. Éste es uno de los servicios por defecto que el framework nos ofrece para trabajar. Su tiempo de vida de éste es **Singleton** por lo que a través de él podemos realizar configuraciones que se reflejarán en toda nuestra aplicación.

Esto lo utilizaremos para enviar un token de autenticación al servidor cuando creemos nuestro sistema de usuarios, más adelante. Con el **HttpClient**, podemos realizar todo tipo de peticiones HTTP incluido el uso de métodos como get, post, input y delete. Estos métodos los usaremos a lo largo de este módulo.

En el componente `FetchData` (que vino con el ejemplo) vemos que se inyectó al **HttpClient** y luego lo utiliza en el **OnInitializedAsync**.

FetchData.razor

```
@page "/fetchdata"
@using BlazorPeliculas.Shared
@inject HttpClient Http

<PageTitle>Weather forecast</PageTitle>

<h1>Weather forecast</h1>

<p>This component demonstrates fetching data from the server.</p>

@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
                <th>Date</th>
                <th>Temp. (C)</th>
                <th>Temp. (F)</th>
                <th>Summary</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var forecast in forecasts)
```

```
{  
    <tr>  
        <td>@forecast.Date.ToShortDateString()</td>  
        <td>@forecast.TemperatureC</td>  
        <td>@forecast.TemperatureF</td>  
        <td>@forecast.Summary</td>  
    </tr>  
}  
}  
</tbody>  
</table>  
}  
  
@code {  
    private WeatherForecast[]? forecasts;  
  
    protected override async Task OnInitializedAsync()  
    {  
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("WeatherForecast");  
    }  
}
```

Con **HttpClient** llama al método **GetFromJsonAsync** que devuelve un array de **WeatherForecast**.

En el proyecto **Server** podemos ver que en Controllers se encuentra la clase **WeatherForecastController.cs** que es el servicio al que estamos consultando desde el proyecto **Client**.

Vale recordar que el proyecto **Client** correrá en el navegador mientras que **Server** en el servidor. Por ello, es necesario hacer consultas HTTP para poder comunicarse desde el navegador contra el servidor.

```
WeatherForecastController.cs  
using BlazorPeliculas.Shared;  
using Microsoft.AspNetCore.Mvc;  
  
namespace BlazorPeliculas.Server.Controllers {  
    [ApiController]  
    [Route("[controller]")]  
    public class WeatherForecastController : ControllerBase {  
        private static readonly string[] Summaries = new []{  
            "Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"  
        };  
  
        private readonly ILogger<WeatherForecastController> _logger;  
  
        public WeatherForecastController(ILogger<WeatherForecastController> logger)  
        {
```

```
_logger = logger;
}

[HttpGet]
public IEnumerable<WeatherForecast> Get()
{
    return Enumerable.Range(1, 5).Select(index => new WeatherForecast
    {
        Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),
        TemperatureC = Random.Shared.Next(-20, 55),
        Summary = Summaries[Random.Shared.Next(Summaries.Length)]
    })
    .ToArray();
}
```

En esta clase vemos que tenemos el método GET que devuelve 5 registros de tipo **WeatherForecast**. El cliente obtendrá dicho array, lo des-serializando el JSON hacia un array de **WeatherForecast** y guardarlo en el campo **forecasts**.

No solo se dispone de llamadas Get. Se pueden hacer `Http.PutFromJsonAsync`, por ejemplo, o `Delete` o incluso `Post`.

También hay métodos como `GetAsync` que nos permite acceder a la cabecera o el estado mientras que los de `FromJson` perdemos toda esa información ya que sólo obtenemos el cuerpo.

Entity Framework Core

Es un ORM (Object Relation Mapping, una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y la utilización de una base de datos relacional como motor de persistencia) para comunicarnos entre C# y la BD. Así como también poder crearla a partir de código. Es decir, va a tener un conjunto de clases llamadas entidades a partir de las cuales se van a crear las tablas de mi base de datos.

En este curso vamos a trabajar con SQL Server, sin embargo, este framework Core es capaz de trabajar con distintos motores de bases de datos como MySQL, Oracle, PostgreSQL, entre otros.

Es multiplataforma: Windows, Linux, MacOS.

No instalaremos EFC en el proyecto Client, porque lo usaremos para conectarnos con la BD y no queremos que se pueda filtrar al cliente el string de conexión, clave u origen de los datos.

Eso es información sensible que yo quiero que se mantenga protegida en mi servidor, así que vamos a trabajar prácticamente exclusivamente con el proyecto Server.

En el proyecto **Server**, click derecho y Manage NuGet packages y buscamos **Microsoft.EntityFrameworkCore**. Elegimos el de SqlServer. También instalamos el de Tools (porque estamos trabajando con Visual Studio). Si usáramos Visual Studio Code, habría que instalar el Design en su lugar.

DbContext

A través de él vamos a indicar qué clases de mi aplicación van a representar tablas, es decir, qué clases son entidades.

Creamos la clase **ApplicationDbContext.cs** en el proyecto Server y la hacemos heredar de **DbContext**. Marcamos **Ctrl+.** para poder agregar el using de **Microsoft.EntityFrameworkCore**. para generar el constructor de:

```
public ApplicationDbContext(DbContextOptions options) : base(options)
```

ApplicationDbContext.cs

```
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : DbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {
        }
    }
}
```

Abrimos el json de appsettings de desarrollo y agregamos el string de conexión:

Appsettings.Development.json

```
{
    "ConnectionStrings": {
        "DefaultConnection": {
            "Server=(localdb)\\MSSQLLocalDB;Database=BlazorPeliculas;Trusted_Connection=True"
        }
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    }
}
```

Abrimos el **Program.cs** de nuestro proyecto **Server** y agregamos el siguiente código luego de que el builder ya esté seteado pero antes de el builder.Build():

Program.cs

```
using BlazorPeliculas.Server;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer("name=DefaultConnection"));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

DBSet

Con éste método indicamos que queremos crear una tabla a partir de una clase. Con el método **Set** se crea el **DBSet**. Con ese comando estamos indicando que **Genre** es una entidad y que generaremos una tabla a partir de ella.

Recordamos que en la entidad de Movie tenemos un listado de **GenresMovie**:

Movie.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public int ID { get; set; }
        [Required]
        public string Title { get; set; } = null!;
        public string? Summary { get; set; }
        public bool OnBillboard { get; set; }
        public string? Trailer { get; set; }
        public DateTime? ReleaseDate { get; set; }
        public string? Poster { get; set; }
        public List<GenresMovie> GenresMovie { get; set; } = new List<GenresMovie>();
        public string? TrimmedTitle {
            get {
                if(string.IsNullOrWhiteSpace(Title)) {
                    return null;
                }

                if(Title.Length > 60) {
                    return Title.Substring(0, 60) + "...";
                }
                else {
                    return Title;
                }
            }
        }
    }
}
```

Esto se conoce como una propiedad de navegación, porque me permite desde una entidad navegar a otra entidad que en este caso va a ser la entidad **GenresMovie**. Ésta va a ser una entidad porque necesitamos una tabla en la cual guardaremos la relación muchos a muchos, entre géneros y películas.

También es importante que desde la entidad de **GenresMovie** coloquemos una propiedad de navegación hacia **Movie**. La propiedad de navegación, como su nombre lo indica, me va a permitir, por ejemplo, a partir de una película, cargar sus géneros a través la propiedad **GenresMovie** que tenemos en **Movie**.

Por eso se llama propiedad de navegación, porque me permite navegar entre entidades. Así, desde **Movie** yo voy a poder navegar hacia la entidad **GenresMovie**.

Y claro, cuando yo digo navegar hacia la entidad **GenresMovie**, me refiero a que si yo tengo una **Movie**, yo voy a poder cargar sus géneros de una manera muy sencilla, pero solamente los géneros de esa película, no los géneros de otras películas.

Entonces también podemos crear en **GenresMovie** propiedades de navegación que vayan desde el **GenresMovie** hacia **Movie** y también desde **GenresMovie** hacia **Genre**.

Eso es importante porque por ejemplo desde **Movie** yo voy a cargar este listado de **GenresMovie**, pero ¿qué tenemos en **GenresMovie** del género? Pues el ID, pero yo quiero el nombre. Quizás yo no quiera solamente el ID del género sino el nombre.

Por eso es importante que yo ahora venga a **GenresMovie** y agregue una propiedad de navegación hacia **Genre** para poder tener el nombre. Lo mismo hacia la película.

GenresMovie.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class GenresMovie {
        public int MovieID { get; set; }
        public int GenreID { get; set; }
        public Genre? Genre { get; set; }
        public Movie? Movie { get; set; }
    }
}
```

Y ya con eso estamos creando una propiedad de navegación hacia **Genre**. Y cuando lleguemos a **GenresMovie**, pues vamos a poder obtener su nombre. Entonces la cadena, es desde **Movie** yo voy a ir a **GenresMovie** y desde ahí a **Genre** y a partir de ahí ya yo voy a tener el nombre del género.

Genre.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Genre {
        public int ID { get; set; }

        [Required(ErrorMessage = "The field '{0}' is required.")]
        public string Name { get; set; } = null!;
        public List<GenresMovie> GenresMovie { get; set; } = new List<GenresMovie>();
    }
}
```

Veamos la lógica en la nomenclatura. Tenemos, por ejemplo, **MovieID** y **Movie**. Esto no es casualidad.

Esto es lo que se conoce como una convención de Entity Framework que me permite a partir de cómo escribamos el código, se van a realizar configuraciones a nivel de la base de datos.

Por ejemplo, al yo decir aquí que esta es una propiedad de navegación **Movie** y sabemos que es una propiedad de navegación porque estamos colocando aquí una entidad y el nombre de esa propiedad de navegación es **Movie**.

Ahora tenemos **MovieID** indicándole a Entity Framework Core que esto se trata de una llave foránea que va a apuntar hacia la tabla que se corresponda con la entidad **Movie**, es decir, que va a apuntar a una llave primaria de esta tabla que se va a corresponder con esta entidad.

Entonces, esta nomenclatura de **Genre** y **GenreID** o **Movie** y **MovieID** no es casualidad.

Sin embargo, no hemos terminado de configurar **GenresMovie**. En las otras entidades, tenemos el parámetro ID que será tomado como PK. Pero en **GenresMovie** no la tenemos ya que querremos que la PK sea una clave compuesta: **MovieID+GenreID**. Esto se consigue con el código de **OnModelCreating**.

ApplicationDbContext.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : DbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder); //No se puede eliminar esta línea.

            modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });

        }

        public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase
        public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
        public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase
        public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla
        GenresMovie a partir de la clase GenresMovie.
    }
}
```

Haremos lo mismo con Actores -> Películas (muchos a muchos) y viceversa (también muchos a muchos).

Creamos el resto de las entidades:

Movie.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public int ID { get; set; }
        [Required]
        public string Title { get; set; } = null!;
        public string? Summary { get; set; }
        public bool OnBillboard { get; set; }
        public string? Trailer { get; set; }
        public DateTime? ReleaseDate { get; set; }
        public string? Poster { get; set; }
    }
}
```

```
public List<GenresMovie> Genres { get; set; } = new List<GenresMovie>();
public List<MovieActor> MovieActor { get; set; } = new List<MovieActor>();
public string? TrimmedTitle {
    get {
        if(string.IsNullOrWhiteSpace(Title)) {
            return null;
        }

        if>Title.Length > 60) {
            return Title.Substring(0, 60) + "...";
        }
        else {
            return Title;
        }
    }
}
```

Actor.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Actor {
        public int ID { get; set; }
        [Required]
        public string Name { get; set; } = null!;
        public string? Bio { get; set; }
        public string? Photo { get; set; }
        public DateTime? BirthDate { get; set; }
        public List<MovieActor> MovieActor { get; set; } = new List<MovieActor>();

        public override bool Equals(object? obj) {
            if(obj is Actor a2) {
                //El objeto recibido es un actor
                return ID == a2.ID;
            }

            return false;
        }
        public override int GetHashCode() {
            return base.GetHashCode();
        }
    }
}
```

MovieActor.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class MovieActor {
        public int ActorID { get; set; }
        public int MovieID { get; set; }
        public Actor? Actor { get; set; }
        public Movie? Movie { get; set; }
        public string? Character { get; set; }
        public int Orden { get; set; }      //El orden de un actor en una película
    }
}
```

VoteMovie.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class VoteMovie {
        public int ID { get; set; }
        public int Voto { get; set; }
        public DateTime VoteWhen { get; set; }
        public int MovieID { get; set; }
        public Movie? Movie { get; set; }
    }
}
```

ApplicationDbContext.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : DbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder);  //No se puede eliminar esta línea.
        }
    }
}
```

```

modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });
modelBuilder.Entity<MovieActor>().HasKey(x => new { x.ActorID, x.MovieID });
}

public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase
Genre.
public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase
Movie.
public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla
GenresMovie a partir de la clase GenresMovie.
public DbSet<MovieActor> MoviesActors => Set<MovieActor>(); //Creamos la tabla MoviesActors
a partir de la clase MovieActor.
}
}

```

Migraciones

Ahora crearemos las migraciones. Una migración se refiere a los cambios que van a ocurrir en la base de datos antes de que ocurran. Es decir, hicimos estos cambios acá en el **ApplicationDbContext**, donde indicamos que, por ejemplo, se va a crear una tabla a partir de la clase género.

Sin embargo, antes de que creemos la tabla de géneros, vamos a crear una clase que es una migración, la cual va a contener el hecho de que en nuestra base de datos se va a crear la tabla de géneros, es decir, una migración, no es más que un paso intermedio entre las configuraciones de mi aplicación con lo que respecta a Entity Framework Core y los cambios que van a ocurrir en nuestra base de datos, es decir, la migración es como esa clase que te va a listar paso por paso.

Abrimos **Tools -> NuGet Package Manager -> Package Manager Console**. Veremos los comandos a correr tanto en el PMC como en el CLI de **Entity Framework Core**.

PMC	EFC cli
Add-Migration Inicial	dotnet ef migrations add Inicial

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

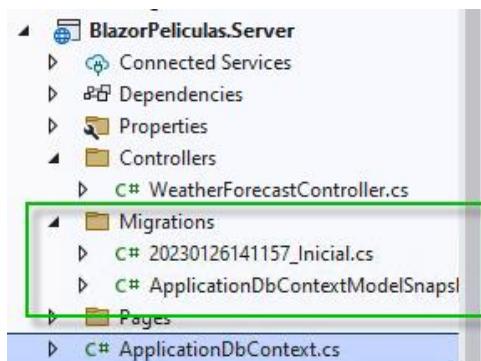
Package Manager Console Host Version 6.4.0.111

Type 'get-help NuGet' to see all available NuGet commands.

```

PM> Add-Migration Inicial
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>

```



Donde **Inicial** es el nombre único (no se puede repetir) de la migración. Vemos que en el proyecto Server se creó la carpeta **Migrations** que es donde se van a colocar nuestras migraciones. Veremos que tenemos la clase parcial **Inicial** y tenemos este método **Up**.

Éste es el que indica los cambios que esta migración va a hacer en la base de datos. Por ejemplo, fíjate que lo primero que va a hacer es crear una tabla llamada **Actors** y esa tabla va a tener las columnas ID, Name, Bio, Photo y BirthDate. también dice que la llave primaria se va a llamar **PK_Actors** y le va a corresponder a la columna **ID**.

Tenemos la tabla **GenresMovie** que va a tener dos columnas **MovieID** y **GenreID**. También dice que la llave primaria va a ser la composición de **GenreID** y **MovieID**. No solo eso también dice que tendremos llaves foráneas que van a ser la que apunta a la tabla principal **Genres** y la que apunta a la tabla principal **Movies**. Y claro, esas son **GenreID** y **MovieID** respectivamente. También se ven los índices que se crearían justo antes de terminar el método **Up**.

20230126141157_Inicial.cs

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace BlazorPeliculas.Server.Migrations {
    /// <inheritdoc />
    public partial class Inicial : Migration {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder) {
            migrationBuilder.CreateTable(
                name: "Actors",
                columns: table => new {
                    ID = table.Column<int>(type: "int", nullable: false)
                        .Annotation("SqlServer:Identity", "1, 1"),
                    Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
                    Bio = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    Photo = table.Column<string>(type: "nvarchar(max)", nullable: true),
                    BirthDate = table.Column<DateTime>(type: "datetime2", nullable: true)
                },
                constraints: table => {
                    table.PrimaryKey("PK_Actors", x => x.ID);
                }
            );
            migrationBuilder.CreateTable(
                name: "Genres",
                columns: table => new {
```

```

ID = table.Column<int>(type: "int", nullable: false)
    .Annotation("SqlServer:Identity", "1, 1"),
Name = table.Column<string>(type: "nvarchar(max)", nullable: false)
},
constraints: table => {
    table.PrimaryKey("PK_Genres", x => x.ID);
});

migrationBuilder.CreateTable(
    name: "Movies",
    columns: table => new {
        ID = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Title = table.Column<string>(type: "nvarchar(max)", nullable: false),
        Summary = table.Column<string>(type: "nvarchar(max)", nullable: true),
        OnBillboard = table.Column<bool>(type: "bit", nullable: false),
        Trailer = table.Column<string>(type: "nvarchar(max)", nullable: true),
        ReleaseDate = table.Column<DateTime>(type: "datetime2", nullable: true),
        Poster = table.Column<string>(type: "nvarchar(max)", nullable: true)
    },
    constraints: table => {
        table.PrimaryKey("PK_Movies", x => x.ID);
    });
}

migrationBuilder.CreateTable(
    name: "GenresMovie",
    columns: table => new {
        MovieID = table.Column<int>(type: "int", nullable: false),
        GenreID = table.Column<int>(type: "int", nullable: false)
    },
    constraints: table => {
        table.PrimaryKey("PK_GenresMovie", x => new { x.GenreID, x.MovieID });
        table.ForeignKey(
            name: "FK_GenresMovie_Genres_GenreID",
            column: x => x.GenreID,
            principalTable: "Genres",
            principalColumn: "ID",
            onDelete: ReferentialAction.Cascade);
        table.ForeignKey(
            name: "FK_GenresMovie_Movies_MovieID",
            column: x => x.MovieID,
            principalTable: "Movies",
            principalColumn: "ID",
            onDelete: ReferentialAction.Cascade);
    });
);

migrationBuilder.CreateTable(
    name: "MoviesActors",
    columns: table => new {

```

```

ActorID = table.Column<int>(type: "int", nullable: false),
MovieID = table.Column<int>(type: "int", nullable: false),
Character = table.Column<string>(type: "nvarchar(max)", nullable: true),
Orden = table.Column<int>(type: "int", nullable: false)
},
constraints: table => {
    table.PrimaryKey("PK_MoviesActors", x => new { x.ActorID, x.MovieID });
    table.ForeignKey(
        name: "FK_MoviesActors_Actors_ActorID",
        column: x => x.ActorID,
        principalTable: "Actors",
        principalColumn: "ID",
        onDelete: ReferentialAction.Cascade);
    table.ForeignKey(
        name: "FK_MoviesActors_Movies_MovieID",
        column: x => x.MovieID,
        principalTable: "Movies",
        principalColumn: "ID",
        onDelete: ReferentialAction.Cascade);
});

migrationBuilder.CreateIndex(
    name: "IX_GenresMovie_MovieID",
    table: "GenresMovie",
    column: "MovieID");

migrationBuilder.CreateIndex(
    name: "IX_MoviesActors_MovieID",
    table: "MoviesActors",
    column: "MovieID");
}

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder) {
    migrationBuilder.DropTable(
        name: "GenresMovie");
    migrationBuilder.DropTable(
        name: "MoviesActors");
    migrationBuilder.DropTable(
        name: "Genres");
    migrationBuilder.DropTable(
        name: "Actors");
    migrationBuilder.DropTable(
        name: "Movies");
}
}

```

Por otro lado, el método **Down** realizaría los **DROP TABLE** para borrar las tablas. Que sirve por si queremos borrar una migración.

Antes de poder aplicar la migración tenemos que crear una BD a partir de esta migración. Para ello, ejecutamos el comando correspondiente en el PMC.

PMC	EFC cli
Update-Database	dotnet ef database update

```

PM> Update-Database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (205ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
        CREATE DATABASE [BlazorPeliculas];
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (68ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
        IF SERVERPROPERTY('EngineEdition') <> 5
        BEGIN
            ALTER DATABASE [BlazorPeliculas] SET READ_COMMITTED_SNAPSHOT ON;
        END;
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (6ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        CREATE TABLE [__EFMigrationsHistory] (
            [MigrationId] nvarchar(150) NOT NULL,
            [ProductVersion] nvarchar(32) NOT NULL,
            CONSTRAINT [PK__EFMigrationsHistory] PRIMARY KEY ([MigrationId])
        );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'__EFMigrationsHistory');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT [MigrationId], [ProductVersion]
        FROM [__EFMigrationsHistory]
        ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20230126141157_Inicial'.
Applying migration '20230126141157_Inicial'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        CREATE TABLE [Actors] (
            [ID] int NOT NULL IDENTITY,
            [Name] nvarchar(max) NOT NULL,
            [Bio] nvarchar(max) NULL,
            [Photo] nvarchar(max) NULL,
            [BirthDate] datetime2 NULL,
            CONSTRAINT [PK_Actors] PRIMARY KEY ([ID])
        );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        CREATE TABLE [Genres] (
            [ID] int NOT NULL IDENTITY,
            [Name] nvarchar(max) NOT NULL,
            CONSTRAINT [PK_Genres] PRIMARY KEY ([ID])
        );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        CREATE TABLE [Movies] (
            [ID] int NOT NULL IDENTITY,
            [Title] nvarchar(max) NOT NULL,

```

```
[Summary] nvarchar(max) NULL,
[OnBillboard] bit NOT NULL,
[Trailer] nvarchar(max) NULL,
[ReleaseDate] datetime2 NULL,
[Poster] nvarchar(max) NULL,
CONSTRAINT [PK_Movies] PRIMARY KEY ([ID])
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [GenresMovie] (
    [MovieID] int NOT NULL,
    [GenreID] int NOT NULL,
    CONSTRAINT [PK_GenresMovie] PRIMARY KEY ([GenreID], [MovieID]),
    CONSTRAINT [FK_GenresMovie_Genres_GenreID] FOREIGN KEY ([GenreID]) REFERENCES [Genres]
    ([ID]) ON DELETE CASCADE,
    CONSTRAINT [FK_GenresMovie_Movies_MovieID] FOREIGN KEY ([MovieID]) REFERENCES [Movies]
    ([ID]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [MoviesActors] (
    [ActorID] int NOT NULL,
    [MovieID] int NOT NULL,
    [Character] nvarchar(max) NULL,
    [Orden] int NOT NULL,
    CONSTRAINT [PK_MoviesActors] PRIMARY KEY ([ActorID], [MovieID]),
    CONSTRAINT [FK_MoviesActors_Actors_ActorID] FOREIGN KEY ([ActorID]) REFERENCES [Actors]
    ([ID]) ON DELETE CASCADE,
    CONSTRAINT [FK_MoviesActors_Movies_MovieID] FOREIGN KEY ([MovieID]) REFERENCES [Movies]
    ([ID]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_GenresMovie_MovieID] ON [GenresMovie] ([MovieID]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_MoviesActors_MovieID] ON [MoviesActors] ([MovieID]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20230126141157_Inicial', N'7.0.2');
Done.
PM>
```

Como utilizamos localdb, podemos acceder a [View -> SQL Server Object Explorer](#).

The screenshot shows the SQL Server Object Explorer interface. Under the 'BlazorPeliculas' database, the 'Tables' node is expanded, showing tables such as 'dbo.GenresMovie', 'dbo.Actors', and 'dbo.Movies'. Each table has its columns listed under the 'Columns' node, including primary keys like 'MovielID' and foreign keys like 'GenreID'. Other nodes like 'Keys', 'Constraints', and 'Indexes' are also visible.

Vemos que hay 2 instancias. Expandimos la de MSSQLLocalDB y buscamos la BD **BlazorPeliculas**. Si expandimos las tablas, vemos que están **Actors**, **Genres**, **GenresMovie**, **Movies** y **MoviesActors**.

También podemos ver que la tabla **GenresMovie** tiene las columnas **MovielID** y **GenreID**, las llaves de PK y 2 FK. A su vez, también vemos los índices que la migración dijo que se crearían... ¡creados!

Respecto a la tabla **_EFMigrationsHistory** es básicamente una tabla auxiliar que utiliza **Entity Framework Core** para indicar cual es Migraciones han sido aplicadas en nuestra base de datos.

Si hacemos click-derecho, View Data, veremos que tenemos que se ha aplicado la migración que es la migración inicial y dice que utilizamos Entity Framework 7.02 para esto.

dbo._EFMigr...History [Data] ▾ X 202301261411		
	MigrationId	ProductVersion
▶	b0230126141157_Inicial	7.0.2
*	NULL	NULL

De nuevo, esto es simplemente para que Entity Framework Core sepa que esta migración ya fue aplicada para que así no la aplique dos veces en nuestra base de datos, pero lo importante es ver que ya tenemos nuestra base de datos y podemos continuar trabajando con el desarrollo de nuestra aplicación.

CRUD

Crear géneros

Crearemos el Controller para los géneros en nuestro proyecto **Server** para poder dar de alta géneros en la tabla correspondiente. Para ello, click derecho en la carpeta Controllers del proyecto Server y agregamos una clase que llamaremos **GenresController.cs**.

Un controlador en .NET es una clase encargada de recibir peticiones HTTP. Así, desde nuestro proyecto Client podremos enviar peticiones HTTP hacia el controlador de géneros y a través de éste interactuaremos con la BD.

Para ello, heredamos de ControllerBase y agregamos el using de Microsoft.AspNetCore.Mvc (con Ctrl+.)

GenresController.cs

```
using Microsoft.AspNetCore.Mvc;

namespace BlazorPeliculas.Server.Controllers {
    public class GenresController: ControllerBase {
    }
}
```

Si bien no es necesario, al docente le gusta poner el prefijo api/ en el Route para los controladores que tengan que ver con la webapi. Utilizaremos un constructor porque necesitamos recibir **ApplicationDbContext**. Utilizaremos el sistema de inyección de dependencias para traer una instancia de la **ApplicationDbContext**, porque es a través de ésta que podremos insertar géneros en mi base de datos.

Crearemos el método **POST** para poder dar de alta los géneros con una tarea asincrónica que devolverá **ActionResult** que se trata de una respuesta genérica y recibirá un **Genre** llamado **genre**.

El **SaveChangesAsync** es quien realmente realiza el INSERT en la tabla **Genres**. Como EFC recibe un género y esta entidad de utilizó para realizar la tabla **Genres** (DBSet), sabe que el INSERT es en esa tabla. Podemos devolver Ok() si se hizo correctamente:

GenresController.cs

```
[HttpPost]
public async Task<ActionResult> Post(Genre genre) {
    context.Add(genre); //Marco el género para ser insertado.
    await context.SaveChangesAsync(); //Se hace el INSERT
    return Ok(); //Todo se hizo correctamente
}
```

También se puede devolver un int (el ID creado). El atributo **ApiController**, hace que nuestro controlador se comporte como un controlador de webapi y es lo que va a hacer que nuestro código funcione.

GenresController.cs

```
using Microsoft.AspNetCore.Mvc;

using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            context.Add(genre); //Marco el género para ser insertado.
            await context.SaveChangesAsync(); //Se hace el INSERT
            return genre.ID; //Todo se hizo correctamente
        }
    }
}
```

Ahora usaremos el **HttpClient** para consultar el método recién creado. Para ello, centralizaremos la lógica de realizar peticiones en la clase **Repository**. Así mismo, centralizaremos las respuestas de nuestra webapi para tener un objeto común a través del cual podemos acceder a, por ejemplo, si hubo un error o no en el webapi, cuál fue la respuesta, etc.

Crearemos la clase **HttpResponseWrapper** en la carpeta **Repositorios** del proyecto **Client**. Lo haremos genérico para que la respuesta pueda ser de cualquier tipo.

El campo **httpResponseMessage** lo usaremos en caso de que queramos acceder a toda la información posible que me trae la respuesta HTTP como por ejemplo las cabeceras, etc.

HttpResponseWrapper.cs

```
namespace BlazorPeliculas.Client.Repositories {
    public class HttpResponseWrapper<T> {
        public HttpResponseWrapper(T? response, bool Error, HttpResponseMessage httpResponseMessage) {
            Response = response;
        }
    }
}
```

```

        this.Error = Error;
        this.httpResponseMessage = httpResponseMessage;
    }
    public bool Error { get; set; }
    public T? Response { get; set; }
    public HttpResponseMessage httpResponseMessage { get; set; }
}
}

```

Vamos a nuestro **Repository.cs**. Vemos que hacemos un POST hacia mi URL y luego estoy encapsulado en mi objeto **HttpResponseWrapper** la respuesta de esa petición http.

Repository.cs

```

using BlazorPeliculas.Shared.Entities;
using System.Text;
using System.Text.Json;

namespace BlazorPeliculas.Client.Repositories {
    public class Repository : IRepository {
        public HttpClient HttpClient { get; }

        public Repository(HttpClient httpClient) {
            HttpClient = httpClient;
        }

        public async Task<HttpResponseWrapper<object>> Post<T>(string url, T send) {
            var sendJSON = JsonSerializer.Serialize(send);
            var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
            var responseHttp = await HttpClient.PostAsync(url, sendContent);
            return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
        }

        public List<Movie> GetMovies() {
            return new List<Movie>() {
                new Movie {Title = "Wakanda forever",
                    ReleaseDate = new DateTime(2022, 11, 11),
                    Poster =
                    "https://upload.wikimedia.org/wikipedia/en/thumb/3/3b/Black_Panther_Wakanda_Forever_poster.jpg/220px-Black_Panther_Wakanda_Forever_poster.jpg",
                    new Movie {Title = "Moana",
                        ReleaseDate = new DateTime(2016, 11, 23),
                        Poster =
                        "https://upload.wikimedia.org/wikipedia/en/thumb/2/26/Moana_Teaser_Poster.jpg/220px-Moana_Teaser_Poster.jpg" },
                    new Movie {Title = "Inception",
                        ReleaseDate = new DateTime(2010, 7, 16),
                        Poster =
                        "https://upload.wikimedia.org/wikipedia/en/2/2e/Inception_%282010%29_theatrical_poster.jpg" }
                };
            }
        }
    }
}

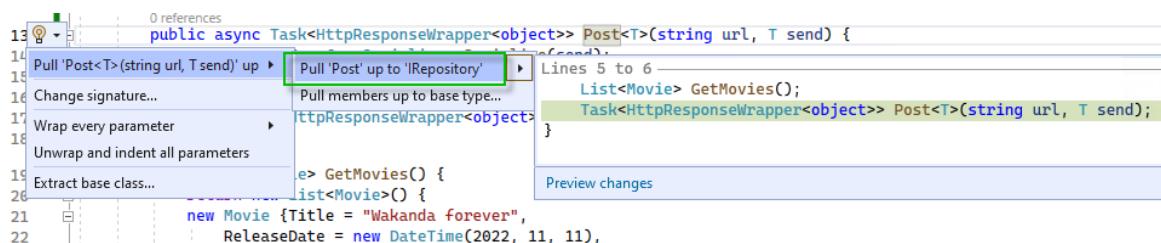
```

```

        }
    }
}

```

Ahora llevaremos la **Task<HttpResponseWrapper> Post** a mi repositorio. Para eso **Ctrl+.** sobre **Post** y hacemos **Pull**.



Y eso simplemente me va a crear la firma en el **IRepository**.

IRepository.cs

```

using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public interface IRepository {
        List<Movie> GetMovies();
        Task<HttpResponseWrapper<object>> Post<T>(string url, T send);
    }
}

```

Un cambio que tenemos que hacer es en la clase **Program**, cambiando el servicio del **IRepository** de **Singleton** a **Scoped**, pues porque en el repositorio estamos utilizando el **httpClient** y éste está como **Scoped**. Para que sean compatibles tenemos que hacer este cambio.

Program.cs

```

using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Repositories;
using CurrieTechnologies.Razor.SweetAlert2;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

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

```

```

void configureServices(IServiceCollection services) {
    services.AddScoped< IRepository, Repository>(); /* Inyectar un IRepository.
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
    clase Repository.*/
    services.AddSweetAlert2();
}
    
```

Hacemos los cambios en CreateGenre para hacer uso de nuestra webapi.

CreateGenre.razor

```

@page "/genres/create"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl

<h3>Create Genre</h3>

<GenreForm @ref="genreForm" Genre="genre" OnValidSubmit="Create" />

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío
    private GenreForm? genreForm;

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private async Task Create() {
        var httpResponse = await repository.Post("api/genres", genre);

        if (httpResponse.Error) {
            await swAl.FireAsync("Error", "An error occurred", SweetAlertIcon.Error);
        }
        else {
            genreForm!.formPostedCorrectly = true;
            navManager.NavigateTo("/genres");
        }
    }
}
    
```

Antes de ejecutar el código, vamos a **SQL Server Object Explorer** elegimos la tabla **Genres** y clickeamos en **View Data** y veremos que la tabla está vacía:

dbo.Genres [Data]		
	ID	Name
*	NULL	NULL

Vamos a crear el género **Comedy** y presionamos **Save changes**.

The screenshot shows a browser window with a sidebar menu for 'BlazorPelículas'. The menu items are: Home, Counter, Fetch data, Movies filter, Genres (which is selected and highlighted in blue), and Actors. The main content area is titled 'Create Genre' and has a form with a 'Name:' input field containing 'Comedy' and a 'Save changes' button. To the right of the browser window is a screenshot of the browser's developer tools Network tab. The tab shows a list of network requests with their names, start times, end times, and durations. One request for 'Genres' is visible, showing a duration of 4ms.

La aplicación nos redirige al índice de géneros lo que indicaría que se hizo correctamente. Vamos a **SQL Server Object Explorer** elegimos la tabla **Genres** y clickeamos en **View Data** y veremos que la tabla YA NO ESTÁ VACÍA.

The screenshot shows the 'Genres' table in the 'dbo' schema. The table has two columns: 'ID' and 'Name'. There is one row with ID 1 and Name 'Comedy'. The table also contains two rows with NULL values for both columns, which are likely part of the primary key definition.

	ID	Name
▶	1	Comedy
*	NULL	NULL

Mensajes de Error

Agregaremos los mensajes de error más descriptivos para poder dar más información respecto al error que ha ocurrido. Eso lo hacemos en nuestra clase de `HttpResponseWrapper` según el código de error.

HttpResponseWrapper.razor

```
using System.Net;
```

```
namespace BlazorPeliculas.Client.Repositories {
    public class HttpResponseWrapper<T> {
        public HttpResponseWrapper(T? response, bool Error, HttpResponseMessage httpResponseMessage) {
            Response = response;
            this.Error = Error;
            this.httpResponseMessage = httpResponseMessage;
        }
        public bool Error { get; set; }
        public T? Response { get; set; }
        public HttpResponseMessage httpResponseMessage { get; set; }

        public async Task<string?> GetErrMsg() {
            if (!Error)
                return null;

            var codStatus = httpResponseMessage.StatusCode;

            if (codStatus == HttpStatusCode.NotFound)
                return "Resource not found";
            else if (codStatus == HttpStatusCode.BadRequest)
                return await httpResponseMessage.Content.ReadAsStringAsync();
            else if (codStatus == HttpStatusCode.Unauthorized)
                return "You need to be logged to do this";
            else if (codStatus == HttpStatusCode.Forbidden)
                return "You don't own the grants needed to perfom this";
            else
                return "An unexpected error has ocurred";
        }
    }
}
```

Ahora, utilizamos el nuevo método desde nuestra CreateGenre.

CreateGenre.razor

```
@page "/genres/create"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl

<h3>Create Genre</h3>

<GenreForm @ref="genreForm" Genre="genre" OnValidSubmit="Create" />

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío
```

```
private GenreForm? genreForm;

//No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
private async Task Create() {
    var httpResponse = await repository.Post("api/genres", genre);

    if (httpResponse.Error) {
        var ErrMessage = await httpResponse.GetErrMsg();
        await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
    }
    else {
        genreForm!.formPostedCorrectly = true;
        navManager.NavigateTo("/genres");
    }
}
```

Para probar esto devolveremos un error de manera forzada:

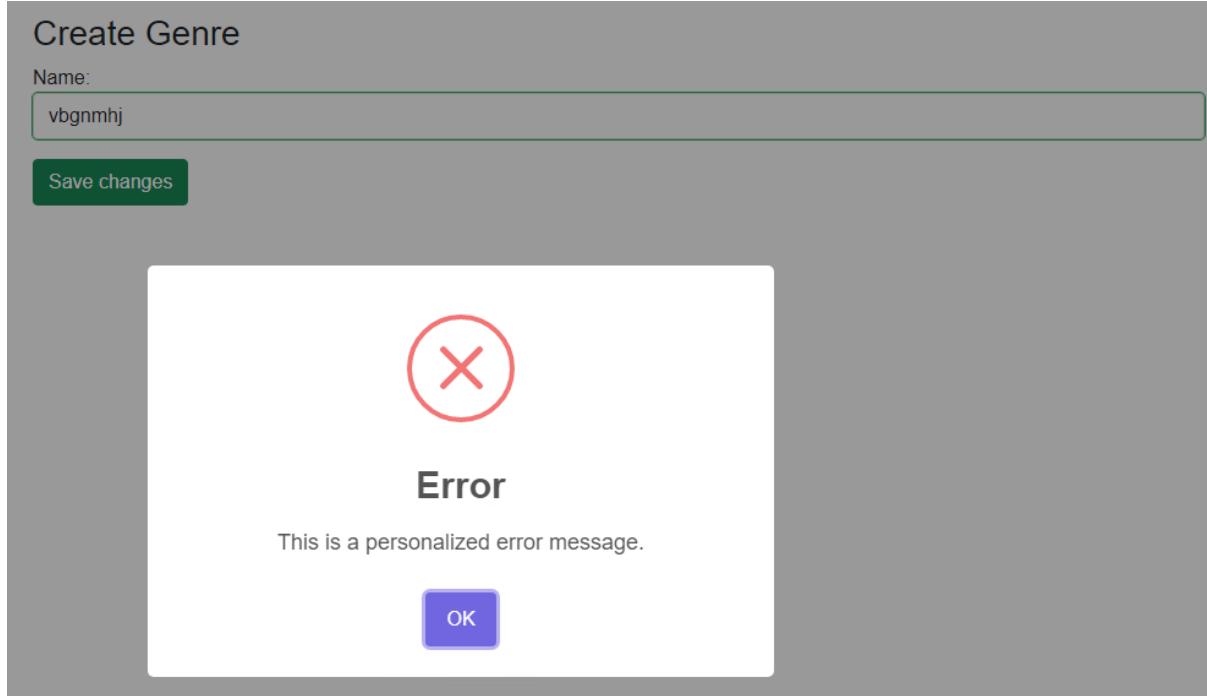
GenresController.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            return BadRequest("This is a personalized error message.");

            context.Add(genre); //Marco el género para ser insertado.
            await context.SaveChangesAsync(); //Se hace el INSERT
            return genre.ID; //Todo se hizo correctamente
        }
    }
}
```



Lanzaremos un mensaje de error genérico porque no quiero compartir el error interno del servidor:

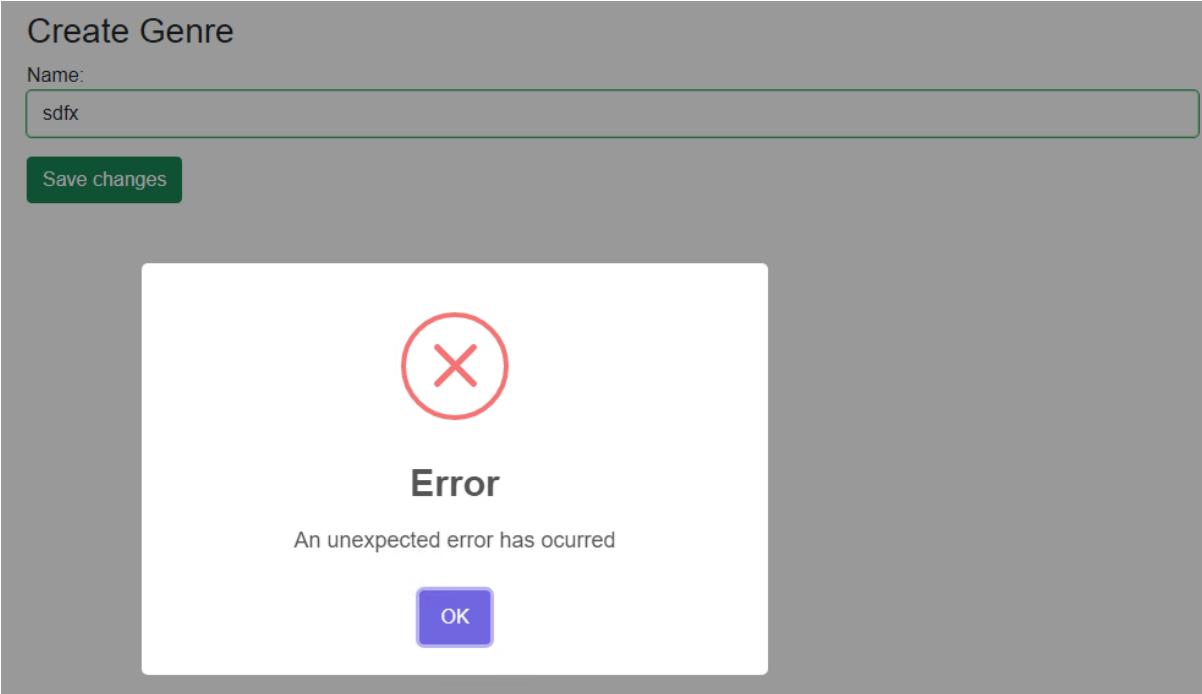
GenresController.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            throw new NotImplementedException();

            context.Add(genre);    //Marco el género para ser insertado.
            await context.SaveChangesAsync(); //Se hace el INSERT
            return genre.ID;      //Todo se hizo correctamente
        }
    }
}
```



Crear actores

Primero crearemos el controlador de actores.

Actors.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;

        public ActorsController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Actor actor) {
            context.Add(actor);
            await context.SaveChangesAsync();
            return actor.ID;
        }
    }
}
```

Y modificamos el [CreateActor](#).

CreateActor.razor

```
@page "/actors/create"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
<h3>Create Actor</h3>

<ActorsForm OnValidSubmit="Create" Actor="Actor" />
@code {
    private Actor Actor = new Actor();

    async Task Create() {
        var httpResponse = await repository.Post("api/actors", Actor);

        if (httpResponse.Error) {
            var ErrMessage = await httpResponse.GetErrMsg();
            await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            navManager.NavigateTo("/actors");
        }
    }
}
```

Probamos.

Create Actor

Name:

Tom Holland

Birthdate:

01/06/1996

Photo

 Tom_Holland...ore.jpg.webp

Bio

Thomas Stanley Holland

He was born 1 June 1996. He is an English actor. His accolades include a British Academy Film Award, three Saturn Awards, a Guinness World Record and an appearance on the Forbes 30 Under 30 Europe list. Some publications have called him one of the most popular actors of his generation.[a]

Holland's career began at age nine when he enrolled in a dancing class, where a choreographer noticed him and arranged for him to audition for a role in *Billy Elliot the Musical* at London's Victoria Palace Theatre. After two years of training, he secured a supporting part in 2008 and was upgraded to the title role that year, which he played until 2010. Holland made his film debut in the disaster drama *The Impossible* (2012) as a teenage tourist trapped in a tsunami, for which he received a London Film Critics Circle Award for Young British Performer of the Year. After this, Holland decided to pursue acting as a full-time career, appearing in *How I Live Now* (2013) and playing historical figures in the film *In the Heart of the Sea* (2015) and the miniseries *Wolf Hall* (2015).

Bio (preview)

Thomas Stanley Holland

He was born 1 June 1996. He is an English actor. His accolades include a British Academy Film Award, three Saturn Awards, a Guinness World Record and an appearance on the Forbes 30 Under 30 Europe list. Some publications have called him one of the most popular actors of his generation.[a]

Holland's career began at age nine when he enrolled in a dancing class, where a choreographer noticed him and arranged for him to audition for a role in *Billy Elliot the Musical* at London's Victoria Palace Theatre. After two years of training, he secured a supporting part in 2008 and was upgraded to the title role that year, which he played until 2010. Holland made his film debut in the disaster drama *The Impossible* (2012) as a teenage tourist trapped in a tsunami, for which he received a London Film Critics Circle Award for Young British Performer of the Year. After this, Holland decided to pursue acting as a full-time career, appearing in *How I Live Now* (2013) and playing historical figures in the film *In the Heart of the Sea* (2015) and the miniseries *Wolf Hall* (2015).

Vemos que tras presionar Save changes, nos vamos al índice de actores por lo que se debería haber grabado correctamente. Si vemos el **View Data** de **Actors**:

	ID	Name	Bio	Photo	BirthDate
	1	Tom Holland	### Thomas St...	UkIGRjYvAABX...	01/06/1996 00:0...
▶*	NULL	NULL	NULL	NULL	NULL

Guardado de imágenes

Para guardar las imágenes crearemos una clase del tipo Interfaz llamada **IFileSaver.cs** en la carpeta **Helpers** del proyecto **Server**. Ésta va a representar un servicio que se encarga de almacenar archivos. Agregaremos las 3 firmas.

SaveFile devolverá un string que es la ruta al archivo. **Container** es una nomenclatura de **Azure** que se corresponde con una carpeta.

Para el caso de **EditFile**, las nuevas versiones de C# permiten implementaciones por defecto. Eso es lo que hacemos en la interfaz para tener que evitar implementar en cada nueva instancia.

```
IFileServer.cs
namespace BlazorPeliculas.Server.Helpers {
    public interface IFileSaver {
        Task<string> SaveFile(byte[] content, string extension, string containerName);
        Task DeleteFile(string path, string containerName);
        async Task<string> EditFile(byte[] content, string extension, string containerName, string path) {
            if(path is not null) {
                await DeleteFile(path, containerName);
            }

            return await SaveFile(content, extension, containerName);
        }
    }
}
```

A

```
ActorsController.cs
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;
```

```
private readonly IFileSaver fileSaver;
private readonly string container = "people";

public ActorsController(ApplicationDbContext context, IFileSaver fileSaver) {
    this.context = context;
    this.fileSaver = fileSaver;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Actor actor) {
    if(!string.IsNullOrWhiteSpace(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actor.Photo = await fileSaver.SaveFile(photoActor, "jpg", container);
    }

    context.Add(actor);
    await context.SaveChangesAsync();
    return actor.ID;
}
}
```

Azure Storage

Implementaremos ahora el código para guardar la imagen en Azure Storage. Básicamente habría que crear un Storage Account (cosa que no hice porque los Storage Accounts son pagos en Azure). Luego de eso, se copia el string de conexión al SA y se lo pega en el **appSettings.Development.json**.

```
appSettings.Development.json
{
    "ConnectionStrings": {
        "DefaultConnection": {
            "Server=(localdb)\\MSSQLLocalDB;Database=BlazorPeliculas;Trusted_Connection=True",
            "AzureStorage": "blablabla"
        },
        "Logging": {
            "LogLevel": {
                "Default": "Information",
                "Microsoft.AspNetCore": "Warning"
            }
        }
    }
}
```

Creamos una clase llamada **FileSaverAzStorage.cs** en la carpeta **Helpers** del proyecto **Server** que implementa la interfaz **IFileSaver**. Agregamos el NuGet Package llamado **Azure.Storage.Blobs** en el proyecto **Server**.

BlobContainerClient es un cliente que me permitirá interactuar con una cuenta de **Azure Storage**.

FileSaverAzStorage.cs

```
using Azure.Storage.Blobs;

namespace BlazorPeliculas.Server.Helpers {
    public class FileSaverAzStorage : IFileSaver {
        private string connectionString;

        public FileSaverAzStorage(IConfiguration configuration) {
            connectionString = configuration.GetConnectionString("AzureStorage")!;
        }

        public async Task DeleteFile(string path, string containerName) {
            var client = new BlobContainerClient(connectionString, containerName);
            await client.CreateIfNotExistsAsync();
            var fileName = Path.GetFileName(path);
            var blob = client.GetBlobClient(fileName);
            await blob.DeleteIfExistsAsync();
        }

        public async Task<string> SaveFile(byte[] content, string extension, string containerName) {
            var client = new BlobContainerClient(connectionString, containerName);
            await client.CreateIfNotExistsAsync();
            client.SetAccessPolicy(Azure.Storage.Blobs.Models.PublicAccessType.Blob);

            if(!extension.StartsWith(".")) extension = "." + extension;
            var fileName = $"{Guid.NewGuid()}{extension}";
            var blob = client.GetBlobClient(fileName);

            using (var ms = new MemoryStream(content)) {
                await blob.UploadAsync(ms);
            }

            return blob.Uri.ToString();
        }
    }
}
```

Agregamos el servicio en el **Program.cs** del proyecto **Server** después de setear la variable **builder** pero antes del **builder.Build**. Lo agregamos como **Transient** porque no vamos a estar compartiendo datos con nadie.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
```

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverAzStorage>();
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

Imágenes locales

Lo primero que hacemos es crear la carpeta **wwwroot** en el proyecto **Server**. En esta carpeta servimos archivos estáticos. Debemos ir a la clase **Program.cs** y nos aseguramos de tener **app.UseStaticFiles()**; después del **app.Builder** ya que esto nos permitirá bajar archivos estáticos desde **wwwroot**.

Creamos una clase llamada **FileSaverLocal.cs** en la carpeta **Helpers** del proyecto **Server** que implementa la interfaz **IFileSaver**.

FileSaverLocal.cs

```
namespace BlazorPeliculas.Server.Helpers {
    public class FileSaverLocal : IFileSaver {
        private readonly IWebHostEnvironment env;
        private readonly IHttpContextAccessor httpContextAccessor;

        public FileSaverLocal(IWebHostEnvironment env, IHttpContextAccessor httpContextAccessor) {
            this.env = env;
            this.httpContextAccessor = httpContextAccessor;
        }

        public Task DeleteFile(string path, string containerName) {
            var fileName = Path.GetFileName(path);
            var folderFileName = Path.Combine(env.WebRootPath, containerName, fileName);

            if(File.Exists(folderFileName))
                File.Delete(folderFileName);

            return Task.CompletedTask;
        }

        public async Task<string> SaveFile(byte[] content, string extension, string containerName) {
            if (!extension.StartsWith(".")) extension = "." + extension;
            var fileName = $"{Guid.NewGuid()}{extension}";
            var folder = Path.Combine(env.WebRootPath, containerName);

            if(!Directory.Exists(folder)) {
                Directory.CreateDirectory(folder);
            }

            string savePath = Path.Combine(folder, fileName);
            await File.WriteAllBytesAsync(savePath, content);

            var actualURL =
${{httpContextAccessor!.HttpContext!.Request.Scheme}://${{httpContextAccessor!.HttpContext.Request.Host}}
;
            //En DBPath tendremos directorio/archivo
            var DBPath = Path.Combine(actualURL, containerName, fileName);
            DBPath = DBPath.Replace("\\", "/");

            return DBPath;
        }
    }
}
```

Cambiamos el servicio para el guarde los archivos localmente. También agregamos el **HttpContextAccessor** que es quién nos permite acceder al contexto para tener el nombre del host y si estamos en http o https.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

Al probar la carga de un nuevo actor, vemos que subió el archivo correctamente.

irce > repos > BlazorPeliculas > Server > wwwroot > people

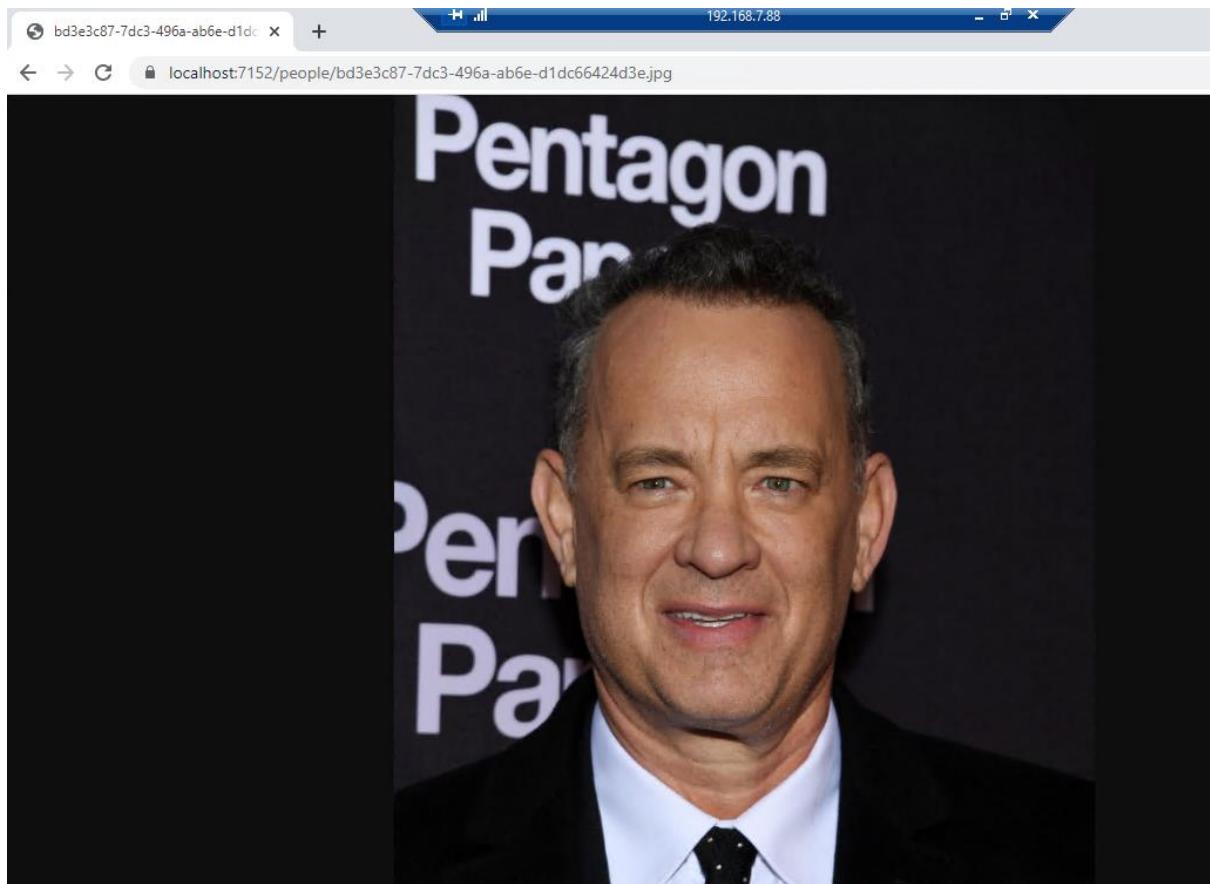


bd3e3c87-7dc3-4
96a-ab6e-d1dc66
424d3e.jpg

Como también que guardó el URL para acceder a la imagen:

ID	Name	Bio	Photo
3	Tom Holland	### Thomas St...	UkIGRjYvAABXRUJQVIA4WAoAAAAGAAAA2wAARAEASUNDUwCAAAAAAIMbGNtcwlQ
4	Tom Hanks	### Thomas Jef...	https://localhost:7152/people/bd3e3c87-7dc3-496a-ab6e-d1dc66424d3e.jpg
*	NULL	NULL	NULL

Y que el archivo es accesible correctamente con dicho URL:



Crear películas

Haremos un cambio en el formulario de películas. Cambiaremos el método a utilizar **OnValidSubmit** por **OnDataAnnotationsValidated**. La idea es tener básicamente un método que se va a encargar de armar los datos de la película que vamos a enviar al servidor, porque recordemos aquí tenemos dos componentes especiales el selector múltiple (géneros) y el selector múltiple typeahead (actores). Lo que necesitamos es recoger de estos componentes la selección del usuario y pasar esa selección a mi controlador de películas.

Recordemos que en **Selected** tenemos un listado de **MultipleSelectorModel** en donde yo tengo el **ID** de cada elemento seleccionado como **Key** y el nombre del género como **Value**. En este caso, tomaremos el **Key** que le **ID** de cada género seleccionado. Por lo tanto, extraeremos esos IDs con el método **Select** para proyectar hacia el tipo de dato **GenresMovie** y le voy a pasar **GenreID** igual a **x.Key**.

Antes de seguir con la selección de los actores. Éstos tienen un personaje en cada película. En el formulario de películas no tenemos (aún) para cargar el personaje. Este dato no debería cargarse en la tabla **Actors** (porque no es un

atributo propio del actor sino del papel que hace en esa película) sería bueno poder tenerlo como campo en la entidad **Actor** (para poder trabajar con el dato) pero que el **Entity Framework Core** no lo interprete como un campo a agregar en nuestra base de datos. Esto se logra con el atributo **NotMapped**.

Actor.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Actor {
        public int ID { get; set; }
        [Required]
        public string Name { get; set; } = null!;
        public string? Bio { get; set; }
        public string? Photo { get; set; }
        public DateTime? BirthDate { get; set; }
        [NotMapped]
        public string? Character { get; set; }
        public List<MovieActor> MovieActor { get; set; } = new List<MovieActor>();

        public override bool Equals(object? obj) {
            if(obj is Actor a2) {
                //El objeto recibido es un actor
                return ID == a2.ID;
            }

            return false;
        }
        public override int GetHashCode() {
            return base.GetHashCode();
        }
    }
}
```

Con esto conseguimos que en nuestras variables del tipo **Actor** tengamos el campo **Character** disponible pero que no se guarde en la tabla **Actors**.

Ahora sí podemos agregar en el **RenderFragment** de **ListTemplate** un **input** del tipo **text** para ingresar el personaje que lo binderemos con **Actor.Character** para que se guarde en dicho campo miembro.

MovieForm.cs

```
<EditForm Model="Movie" OnValidSubmit="OnDataAnnotationsValidated">
    <DataAnnotationsValidator />

    <div class="mb-3">
        <label>Title:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Movie.Title" />
            <ValidationMessage For="@(() => Movie.Title)" />
        </div>
    </div>

    <div class="mb-3">
        <label>On billboard:</label>
        <div>
            <InputCheckbox @bind-Value="@Movie.OnBillboard" />
            <ValidationMessage For="@(() => Movie.OnBillboard)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Release date:</label>
        <div>
            <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
            <ValidationMessage For="@(() => Movie.ReleaseDate)" />
        </div>
    </div>

    <div class="mb-3">
        <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
    </div>

    <div class="mb-3 form-markdown">
        <InputMD @bind-Value="@Movie.Summary"
            For="@(() => Movie.Summary)"
            Label="Summary" />
    </div>

    <div class="mb-3">
        <label>Genres:</label>
        <div>
            <MultipleSelector Selected="Selected" Unselected="Unselected" />
        </div>
    </div>

    <div class="mb-3">
        <label>Actors:</label>
        <div>
            <MultipleSelectorTypeahead Context="Actor" SearchMethod="SearchActors"
                SelectedElements="SelectedActors" NotFoundTemplate="No actors were found" />
        </div>
    </div>
```

```

<ListTemplate>
    @Actor.Name / <input type="text" placeholder="Character" @bind="Actor.Character" />
</ListTemplate>
<ResultTemplate>
    
    @Actor.Name
</ResultTemplate>
</MultipleSelectorTypeahead>
</div>
</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    [Parameter]
    public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
    [Parameter]
    [EditorRequired]
    public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();

    [Parameter]
    public List<Actor> SelectedActors { get; set; } = new List<Actor>();

    private List<MultipleSelectorModel> Selected { get; set; } = new List<MultipleSelectorModel>();
    private List<MultipleSelectorModel> Unselected { get; set; } = new List<MultipleSelectorModel>();

    string? ImageURL;

    protected override void OnInitialized() {
        if(!string.IsNullOrEmpty(Movie.Poster)) {
            ImageURL = Movie.Poster;
            Movie.Poster = null;      //Al editar, seteamos el URL y limpiamos el dato.
                                    //Si no se le carga uno nuevo, no será re-enviado.
        }
    }

    Selected = SelectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(), x.Name)).ToList();
    Unselected = UnselectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(),
x.Name)).ToList();
}

private void SelectedImage(string imageBase64) {
    Movie.Poster = imageBase64;
    ImageURL = null;
}

```

```

}

private async Task<IEnumerable<Actor>> SearchActors(string searchText) {
    //Por el momento va hardcodeado...
    return new List<Actor>() {
        new Actor {ID=1, Name="Tom Holland", Photo =
"https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Tom_Holland_by_Gage_Skidmore.jpg/1200px-Tom_Holland_by_Gage_Skidmore.jpg"},
        new Actor {ID=2, Name="Tom Hanks", Photo =
"https://upload.wikimedia.org/wikipedia/commons/a/a9/Tom_Hanks_TIFF_2019.jpg"}
    };
}

private async Task OnDataAnnotationsValidated() {
    //Obtenemos el listado de géneros seleccionados recuperando los IDs (Key).
    Movie.GenresMovie = SelectedGenres
        .Select(x => new GenresMovie { GenreID = int.Parse(x.Key) }).ToList();

    //Obtenemos el listado de actores seleccionados recuperando los IDs (ID) y los personajes de c/u.
    Movie.MovieActor = SelectedActors
        .Select(x => new MovieActor { ActorID = x.ID, Character = x.Character }).ToList();

    await OnValidSubmit.InvokeAsync();
}
}

```

Ya estaríamos en condiciones de invocar la webapi desde **CreateMovie** pero al finalizar la grabación necesitaríamos el ID de la misma. Creamos un nuevo método **Post** que devuelva un **TResponse** para lo cual crearemos las opciones por defecto de Json:

Repository.cs

```

using BlazorPeliculas.Shared.Entities;
using System.Text;
using System.Text.Json;

namespace BlazorPeliculas.Client.Repositories {
    public class Repository : IRepository {
        public HttpClient HttpClient { get; }

        public Repository(HttpClient httpClient) {
            HttpClient = httpClient;
        }

        private JsonSerializerOptions DefaultJsonOptions = new JsonSerializerOptions {
            PropertyNameCaseInsensitive = true
        };

        public async Task<HttpResponseWrapper<object>> Post<T>(string url, T send) {
            var sendJSON = JsonSerializer.Serialize(send);

```

```

var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
var responseHttp = await HttpClient.PostAsync(url, sendContent);
return new HttpResponseMessage<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
}
public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send) {
    var sendJSON = JsonSerializer.Serialize(send);
    var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
    var responseHttp = await HttpClient.PostAsync(url, sendContent);

    if(responseHttp.IsSuccessStatusCode) {
        var response = await DeserializeResponse<TResponse>(responseHttp, DefaultJsonOptions);
        return new HttpResponseMessage<TResponse>(response, false, responseHttp);
    }
    return new HttpResponseMessage<TResponse>(default, !responseHttp.IsSuccessStatusCode,
    responseHttp);
}

private async Task<T> DeserializeResponse<T>(HttpResponseMessage httpResponse,
JsonSerializerOptions jsonSerializerOptions) {
    var responseString = await httpResponse.Content.ReadAsStringAsync();
    return JsonSerializer.Deserialize<T>(responseString, jsonSerializerOptions);
}

public List<Movie> GetMovies() {
    return new List<Movie>() {
        new Movie {Title = "Wakanda forever",
            ReleaseDate = new DateTime(2022, 11, 11),
            Poster =
"https://upload.wikimedia.org/wikipedia/en/thumb/3/3b/Black_Panther_Wakanda_Forever_poster.jpg/220px-Black_Panther_Wakanda_Forever_poster.jpg",
            new Movie {Title = "Moana",
                ReleaseDate = new DateTime(2016, 11, 23),
                Poster =
"https://upload.wikimedia.org/wikipedia/en/thumb/2/26/Moana_Teaser_Poster.jpg/220px-Moana_Teaser_Poster.jpg" },
            new Movie {Title = "Inception",
                ReleaseDate = new DateTime(2010, 7, 16),
                Poster =
"https://upload.wikimedia.org/wikipedia/en/2/2e/Inception_%282010%29_theatrical_poster.jpg" }
        };
    }
}
}
    
```

Hacemos un Pull para agregar la firma a nuestra interfaz **IRepository**.

IRepository.cs

```

using BlazorPeliculas.Shared.Entities;
namespace BlazorPeliculas.Client.Repositories {
    public interface IRepository {
        List<Movie> GetMovies();
    }
}
    
```

```

        Task<HttpResponseWrapper<object>> Post<T>(string url, T send);
        Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send);
    }
}
    
```

Ahora sí estamos en condiciones de recibir el ID de la película creada y hacer el **NavigateTo** correspondiente:

CreateForm.cs

```

@page "/movies/create"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
<h3>Create Movie</h3>
<MoviesForm Movie="Movie" OnValidSubmit="Create"
    UnselectedGenres="Unselected"/>
@code {
    private Movie Movie = new Movie();
    private List<Genre> Unselected = new List<Genre>();

    protected override void OnInitialized() {
        //Por el momento está hardcodeado...
        Unselected = new List<Genre>() {
            new Genre() { ID = 1, Name = "Comedy" },
            new Genre() { ID = 2, Name = "Drama" },
            new Genre() { ID = 3, Name = "Action" },
            new Genre() { ID = 4, Name = "Sci-fi" }
        };
    }

    async Task Create() {
        var httpResponse = await repository.Post<Movie, int>("api/movies", Movie);

        if (httpResponse.Error) {
            var ErrMessage = await httpResponse.GetErrMsg();
            await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            var movieID = httpResponse.Response;
            navManager.NavigateTo($"~/movie/{movieID}/{Movie.Title.Replace(" ", "-")}");
        }
    }
}
    
```

Ya podemos crear el controlador de películas. Agregamos la clase **MoviesController** en la carpeta **Controllers** del proyecto **Server** y lo hacemos heredar de ControllerBase.

MoviesController.cs

```

using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.Entities;
    
```

```
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context, IFileSaver fileSaver) {
            this.context = context;
            this.fileSaver = fileSaver;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Movie movie) {
            if (!string.IsNullOrWhiteSpace(movie.Poster)) {
                var poster = Convert.FromBase64String(movie.Poster);
                movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
            }

            context.Add(movie);
            await context.SaveChangesAsync();
            return movie.ID;
        }
    }
}
```

Agregamos el campo **Trailer** en nuestro formulario de películas.

Movies.cs

```
<EditForm Model="Movie" OnValidSubmit="OnDataAnnotationsValidated">
<DataAnnotationsValidator />

<div class="mb-3">
    <label>Title:</label>
    <div>
        <InputText class="form-control" @bind-Value="@Movie.Title" />
        <ValidationMessage For="@(() => Movie.Title)" />
    </div>
</div>

<div class="mb-3">
    <label>On billboard:</label>
    <div>
        <InputCheckbox @bind-Value="@Movie.OnBillboard" />
        <ValidationMessage For="@(() => Movie.OnBillboard)" />
    </div>
</div>
```

```

        </div>
    </div>

    <div class="mb-3">
        <label>Trailer:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Movie.Trailer" />
            <ValidationMessage For="@(() => Movie.Trailer)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Release date:</label>
        <div>
            <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
            <ValidationMessage For="@(() => Movie.ReleaseDate)" />
        </div>
    </div>

    <div class="mb-3">
        <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
    </div>

    <div class="mb-3 form-markdown">
        <InputMD @bind-Value="@Movie.Summary"
            For="@(() => Movie.Summary)"
            Label="Summary" />
    </div>

    <div class="mb-3">
        <label>Genres:</label>
        <div>
            <MultipleSelector Selected="Selected" Unselected="Unselected"></MultipleSelector>
        </div>
    </div>

    <div class="mb-3">
        <label>Actors:</label>
        <div>
            <MultipleSelectorTypeahead Context="Actor" SearchMethod="SearchActors"
                SelectedElements="SelectedActors" NotFoundTemplate="No actors were found">
                <ListTemplate>
                    @Actor.Name / <input type="text" placeholder="Character" @bind="Actor.Character" />
                </ListTemplate>
                <ResultTemplate>
                    
                    @Actor.Name
                </ResultTemplate>
            </MultipleSelectorTypeahead>
        </div>
    </div>

```

```

</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    [Parameter]
    public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
    [Parameter]
    [EditorRequired]
    public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();

    [Parameter]
    public List<Actor> SelectedActors { get; set; } = new List<Actor>();

    private List<MultipleSelectorModel> Selected { get; set; } = new List<MultipleSelectorModel>();
    private List<MultipleSelectorModel> Unselected { get; set; } = new List<MultipleSelectorModel>();

    string? ImageURL;

    protected override void OnInitialized() {
        if(!string.IsNullOrEmpty(Movie.Poster)) {
            ImageURL = Movie.Poster;
            Movie.Poster = null;      //Al editar, seteamos el URL y limpiamos el dato.
                                    //Si no se le carga uno nuevo, no será re-enviado.
        }

        Selected = SelectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(), x.Name)).ToList();
        Unselected = UnselectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(),
x.Name)).ToList();
    }

    private void SelectedImage(string imageBase64) {
        Movie.Poster = imageBase64;
        ImageURL = null;
    }

    private async Task<IEnumerable<Actor>> SearchActors(string searchText) {
        //Por el momento va hardcodeado...
        return new List<Actor>() {
            new Actor {ID=1, Name="Tom Holland", Photo =
"https://upload.wikimedia.org/wikipedia/commons/thumb/3/3c/Tom_Holland_by_Gage_Skidmore.jpg/1200px-Tom_Holland_by_Gage_Skidmore.jpg"},
            new Actor {ID=2, Name="Tom Hanks", Photo =

```

```

"https://upload.wikimedia.org/wikipedia/commons/a/a9/Tom_Hanks_TIFF_2019.jpg"}
```

```

    };
}

private async Task OnDataAnnotationsValidated() {
    //Obtenemos el listado de géneros seleccionados recuperando los IDs (Key).
    Movie.GenresMovie = Selected
        .Select(x => new GenresMovie { GenreID = int.Parse(x.Key) }).ToList();

    //Obtenemos el listado de actores seleccionados recuperando los IDs (ID) y los personajes de c/u.
    Movie.MovieActor = SelectedActors
        .Select(x => new MovieActor { ActorID = x.ID, Character = x.Character }).ToList();

    await OnValidSubmit.InvokeAsync();
}
}

```

Al hacer la prueba, no podremos seleccionar géneros ni actores porque actualmente ambos listados están hardcodeados. Más adelante, cuando editemos la película, guardaremos esa información.

Create Movie

Title:

On billboard:

Trailer:

Release date:

Poster Spider-Man_FarFromHome_poster.jpg



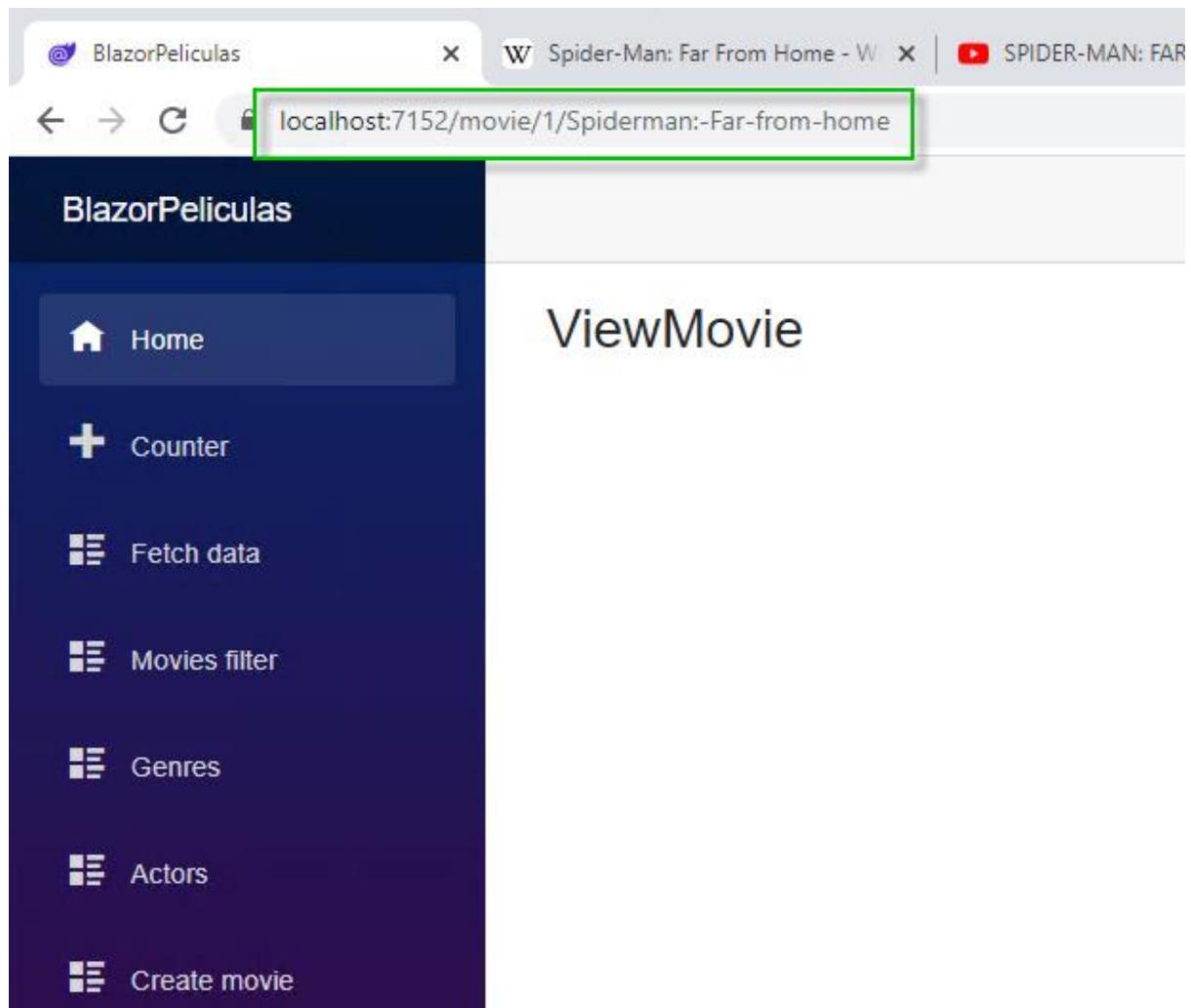
Summary

Summary
In Ixenco, Mexico, Nick Fury and Maria Hill investigate an unnatural storm and encounter the Earth Elemental. Quentin Beck, a super-powered individual, arrives to defeat the creature, and is subsequently recruited by Fury and Hill. In New York City.

Summary (preview)

Summary
In Ixenco, Mexico, Nick Fury and Maria Hill investigate an unnatural storm and encounter the Earth Elemental. Quentin Beck, a super-powered individual, arrives to defeat the creature,

Al presionar grabar nos vamos a la página de visualización de la película (que aún no muestra info de la misma):



Y también la vemos grabada en la BD:

ID	Title	Summary	OnBillboard	Trailer	ReleaseDate	Poster
1	Spiderman: Far ...	### Summary...	True	Nt9L1jCKGnE	26/06/2019 00:0...	https://localho...
*	NULL	NULL	NULL	NULL	NULL	NULL

Leyendos registros

Iremos a nuestro **Repository** para hacer los métodos **GET** que nos permita obtener datos.

Repository.cs

```
using BlazorPeliculas.Shared.Entities;
using System.Text;
using System.Text.Json;

namespace BlazorPeliculas.Client.Repositories {
    public class Repository : IRepository {
        public HttpClient HttpClient { get; }

        public Repository(HttpClient httpClient) {
            HttpClient = httpClient;
        }

        private JsonSerializerOptions DefaultJsonOptions = new JsonSerializerOptions {
            PropertyNameCaseInsensitive = true
        };

        //Lo haremos con T porque podríamos recibir un listado de géneros, actores, películas, etc.
        public async Task<HttpResponseWrapper<T>> Get<T>(string url) {
            var responseHTTP = await HttpClient.GetAsync(url);
            if (responseHTTP.IsSuccessStatusCode) {
                var response = await DeserializeResponse<T>(responseHTTP, DefaultJsonOptions);
                return new HttpResponseMessageWrapper<T>(response, Error: false, responseHTTP);
            }
            else {
                return new HttpResponseMessageWrapper<T>(default, Error: true, responseHTTP);
            }
        }

        public async Task<HttpResponseWrapper<object>> Post<T>(string url, T send) {
            var sendJSON = JsonSerializer.Serialize(send);
            var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
            var responseHttp = await HttpClient.PostAsync(url, sendContent);
            return new HttpResponseMessageWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
        }

        public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send) {
            var sendJSON = JsonSerializer.Serialize(send);
            var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
            var responseHttp = await HttpClient.PostAsync(url, sendContent);

            if(responseHttp.IsSuccessStatusCode) {
                var response = await DeserializeResponse<TResponse>(responseHttp, DefaultJsonOptions);
                return new HttpResponseMessageWrapper<TResponse>(response, false, responseHttp);
            }
            return new HttpResponseMessageWrapper<TResponse>(default, !responseHttp.IsSuccessStatusCode, responseHttp);
        }

        private async Task<T> DeserializeResponse<T>(HttpResponseMessage httpResponse,
        JsonSerializerOptions jsonSerializerOptions) {
            var responseString = await httpResponse.Content.ReadAsStringAsync();
        }
    }
}
```

```

        return JsonSerializer.Deserialize<T>(responseStting, jsonSerializerOptions);
    }

    public List<Movie> GetMovies() {
        return new List<Movie>() {
            new Movie {Title = "Wakanda forever",
                ReleaseDate = new DateTime(2022, 11, 11),
                Poster =
                "https://upload.wikimedia.org/wikipedia/en/thumb/3/3b/Black_Panther_Wakanda_Forever_poster.jpg/220px-Black_Panther_Wakanda_Forever_poster.jpg",
                new Movie {Title = "Moana",
                    ReleaseDate = new DateTime(2016, 11, 23),
                    Poster =
                    "https://upload.wikimedia.org/wikipedia/en/thumb/2/26/Moana_Teaser_Poster.jpg/220px-Moana_Teaser_Poster.jpg" },
                new Movie {Title = "Inception",
                    ReleaseDate = new DateTime(2010, 7, 16),
                    Poster =
                    "https://upload.wikimedia.org/wikipedia/en/2/2e/Inception_%282010%29_theatrical_poster.jpg" }
            };
        }
    }
}

```

Hacemos el Pull para llevar el nuevo método a la interfaz:

```

IRepository.cs
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public interface IRepository {
        Task<HttpResponseWrapper<T>> Get<T>(string url);
        List<Movie> GetMovies();
        Task<HttpResponseWrapper<object>> Post<T>(string url, T send);
        Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send);
    }
}

```

Antes de trabajar con nuestro listado, haremos un cambio en GenericList. Ya que este muestra los elementos de a uno y, en este caso, queremos mostrar una tabla completa y no los géneros de manera independiente.

```

GenericList.cs
@typeparam Item
@if (List is null) {
    @if (Loading is null) {
        
    }
}

```

```

else {
    @Loading
}
}
else if (List.Count == 0) {
    @if (NoRecords is null) {
        <p>No movies to show</p>
    }
    else {
        @NoRecords
    }
}
else {
    if(HasRecords is not null)
        foreach (var elem in List) {
            @HasRecords(elem)
        }
    else {
        @HasRecordsComplete
    }
}

@code {
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;
    [Parameter]
    public RenderFragment<TItem> HasRecords { get; set; } = null!;
    [Parameter] public RenderFragment HasRecordsComplete { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public List<TItem> List { get; set; } = null!;
}

```

También tenemos que implementar el Get en la controller de géneros:

GenresController.cs

```

using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }
    }
}

```

```

        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Genre>>> Get() {
            return await context.Genres.ToListAsync();
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            context.Add(genre); //Marco el género para ser insertado.
            await context.SaveChangesAsync(); //Se hace el INSERT
            return genre.ID; //Todo se hizo correctamente
        }
    }
}

```

Ahora sí podemos implementar el Get y visualización en el listado de géneros:

GenresList.cs

```

@page "/genres"
@inject IRepository repository

<h3>Genres</h3>

<div class="mb-3">
    <a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

<GenericList List="Genres">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach(var item in Genres!) {
                    <tr>
                        <td>
                            <a class="btn btn-success">Edit</a>
                            <button class="btn btn-danger">Delete</button>
                        </td>
                        <td>@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>

```

```
@code {
    public List<Genre>? Genres { get; set; }

    protected async override Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<Genre>>("api/genres");
        Genres = responseHTTP.Response;
    }
}
```

Probamos el listado:



Name
Comedy

Agregamos un nuevo género:



Name:
Action

Save changes

Al darle salvar, se va al listado de géneros:



Name
Comedy
Action

Hacemos lo mismo con el listado de actores:

ActorsList.cs

```

@page "/actors"
@inject IRepository repository
<h3>Actors list</h3>

<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>

<GenericList List="Actors">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Actors!) {
                    <tr>
                        <td>
                            <a class="btn btn-success">Edit</a>
                            <button class="btn btn-danger">Delete</button>
                        </td>
                        <td>@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>

```

```

@code {
    public List<Actor>? Actors { get; set; }

    protected async override Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<Actor>>("api/actors");
        Actors = responseHTTP.Response!;
    }
}

```

Y agregamos el GET en su controller.

ActorController.cs

```

using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;

```

```

using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly string container = "people";

        public ActorsController(ApplicationDbContext context, IFileSaver fileSaver) {
            this.context = context;
            this.fileSaver = fileSaver;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Actor>>> Get() {
            return await context.Actors.ToListAsync();
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Actor actor) {
            if(!string.IsNullOrWhiteSpace(actor.Photo)) {
                //Nos mandaron una foto desde el frontend
                var photoActor = Convert.FromBase64String(actor.Photo);
                actor.Photo = await fileSaver.SaveFile(photoActor, ".jpg", container);
            }

            context.Add(actor);
            await context.SaveChangesAsync();
            return actor.ID;
        }
    }
}

```

Vemos el listado de actores:

Actors list

[Create actor](#)

		Name
Edit	Delete	Tom Holland
Edit	Delete	Tom Hanks

Vamos a CreateMovie para traer el listado de géneros desde la BD. El primer cambio es que usaremos **OnInitializedAsync** en vez de **OnInitialized**. Otro cambio

que haremos es que no queremos mostrar el formulario hasta no haber recibido el listado de géneros.

CreateMovie.cs

```
@page "/movies/create"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
<h3>Create Movie</h3>

@if(ShowForm) {
<MoviesForm Movie="Movie" OnValidSubmit="Create"
    UnselectedGenres="Unselected"/>
}
else {

}
@code {
    private Movie Movie = new Movie();
    private List<Genre> Unselected = new List<Genre>();
    public bool ShowForm { get; set; } = false;

    protected async override Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<Genre>>("api/genres");
        Unselected = responseHTTP.Response!;
        ShowForm = true;
    }

    async Task Create() {
        var httpResponse = await repository.Post<Movie, int>("api/movies", Movie);

        if (httpResponse.Error) {
            var ErrMessage = await httpResponse.GetErrorMessage();
            await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            var movieID = httpResponse.Response;
            navManager.NavigateTo($""/movie/{movieID}/{Movie.Title.Replace(" ", "-")}");
        }
    }
}
```

Create Movie



Genres:

Comedy
Action

>>

<<

Actors:

Activate Windows
Go to Settings to activate Windows.

Save changes

Filtros

Para el selector múltiple de actores, necesitaremos utilizar filtros. Es decir, agregaremos un endpoint en el controlador de actores para poder filtrar por su nombre. Esto es para poder trabajar con el typeahead de actores. Para ello, hacemos un **Where** cuyo nombre contenga el texto buscado y un **Take** para que sólo traiga 5 registros.

ActorsController.cs

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
```

```
[Route("api/actors"), ApiController]
public class ActorsController: ControllerBase {
    private readonly ApplicationDbContext context;
    private readonly IFileSaver fileSaver;
    private readonly string container = "people";

    public ActorsController(ApplicationDbContext context, IFileSaver fileSaver) {
        this.context = context;
        this.fileSaver = fileSaver;
    }

    [HttpGet]
    public async Task<ActionResult<IEnumerable<Actor>>> Get() {
        return await context.Actors.ToListAsync();
    }

    [HttpGet("search/{searchText}")]
    public async Task<ActionResult<IEnumerable<Actor>>> Get(string searchText) {
        if(string.IsNullOrWhiteSpace(searchText))
            return new List<Actor>();

        searchText = searchText.ToLower();
        return await context.Actors
            .Where(x => x.Name.ToLower().Contains(searchText))
            .Take(5)
            .ToListAsync();
    }

    [HttpPost]
    public async Task<ActionResult<int>> Post(Actor actor) {
        if(!string.IsNullOrWhiteSpace(actor.Photo)) {
            //Nos mandaron una foto desde el frontend
            var photoActor = Convert.FromBase64String(actor.Photo);
            actor.Photo = await fileSaver.SaveFile(photoActor, "jpg", container);
        }

        context.Add(actor);
        await context.SaveChangesAsync();
        return actor.ID;
    }
}
```

Usamos el **GET** en el **MoviesForm**.

MoviesForm.cs

```
@inject IRepository repository

<EditForm Model="Movie" OnValidSubmit="OnDataAnnotationsValidated">
```

```
<DataAnnotationsValidator />

<div class="mb-3">
    <label>Title:</label>
    <div>
        <InputText class="form-control" @bind-Value="@Movie.Title" />
        <ValidationMessage For="@(() => Movie.Title)" />
    </div>
</div>

<div class="mb-3">
    <label>On billboard:</label>
    <div>
        <InputCheckbox @bind-Value="@Movie.OnBillboard" />
        <ValidationMessage For="@(() => Movie.OnBillboard)" />
    </div>
</div>

<div class="mb-3">
    <label>Trailer:</label>
    <div>
        <InputText class="form-control" @bind-Value="@Movie.Trailer" />
        <ValidationMessage For="@(() => Movie.Trailer)" />
    </div>
</div>

<div class="mb-3">
    <label>Release date:</label>
    <div>
        <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
        <ValidationMessage For="@(() => Movie.ReleaseDate)" />
    </div>
</div>

<div class="mb-3">
    <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
</div>

<div class="mb-3 form-markdown">
    <InputMD @bind-Value="@Movie.Summary"
        For="@(() => Movie.Summary)"
        Label="Summary" />
</div>

<div class="mb-3">
    <label>Genres:</label>
    <div>
        <MultipleSelector Selected="Selected" Unselected="Unselected" />
    </div>
</div>
```

```

<div class="mb-3">
    <label>Actors:</label>
    <div>
        <MultipleSelectorTypeahead Context="Actor" SearchMethod="SearchActors"
            SelectedElements="SelectedActors" NotFoundTemplate="No actors were found">
            <ListTemplate>
                @Actor.Name / <input type="text" placeholder="Character" @bind="Actor.Character" />
            </ListTemplate>
            <ResultTemplate>
                
                @Actor.Name
            </ResultTemplate>
        </MultipleSelectorTypeahead>
    </div>
</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    [Parameter]
    public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
    [Parameter]
    [EditorRequired]
    public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();

    [Parameter]
    public List<Actor> SelectedActors { get; set; } = new List<Actor>();

    private List<MultipleSelectorModel> Selected { get; set; } = new List<MultipleSelectorModel>();
    private List<MultipleSelectorModel> Unselected { get; set; } = new List<MultipleSelectorModel>();

    string? imageURL;

    protected override void OnInitialized() {
        if(!string.IsNullOrEmpty(Movie.Poster)) {
            imageURL = Movie.Poster;
            Movie.Poster = null;      //Al editar, seteamos el URL y limpiamos el dato.
                                    //Si no se le carga uno nuevo, no será re-enviado.
        }
    }

    Selected = SelectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(), x.Name)).ToList();
}

```

```

        Unselected = UnselectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(),
x.Name)).ToList();
    }

    private void SelectedImage(string imageBase64) {
        Movie.Poster = imageBase64;
        ImageURL = null;
    }

private async Task<IEnumerable<Actor>> SearchActors(string searchText) {
    var responseHTTP = await repository.Get<List<Actor>>($"api/actors/search/{searchText}");
    return responseHTTP!.Response!;
}

private async Task OnDataAnnotationsValidated() {
    //Obtenemos el listado de géneros seleccionados recuperando los IDs (Key).
    Movie.GenresMovie = Selected
        .Select(x => new GenresMovie { GenreID = int.Parse(x.Key) }).ToList();

    //Obtenemos el listado de actores seleccionados recuperando los IDs (ID) y los personajes de c/u.
    Movie.MovieActor = SelectedActors
        .Select(x => new MovieActor { ActorID = x.ID, Character = x.Character }).ToList();

    await OnValidSubmit.InvokeAsync();
}
}
    
```

Y en el formulario ya está implementando el filtro.

Actors:

The screenshot shows a search input field containing "Tom H". Below it, a list of actors is displayed. The first item is "Tom Holland" with a small profile picture. The second item is "Tom Hanks" with a small profile picture. To the right of the list, there is a blue button labeled "Activate Windows" and a link "Go to Settings to activate Windows".

Crearemos ahora un endpoint para traer el listado de las películas. Como nuestro endpoint devolverá 2 listados, usaremos un **DTO**. Con un **DTO** podemos tener una clase que va a contener datos y esto nos sirve para poner esos datos en un lugar y pasarlo a otras clases. Creamos una carpeta **DTOs** en el proyecto **Shared**. En ella crearemos una clase **pública** llamada **HomePageDTO.cs**.

```

HomePageDTO.cs
@inject IRepository repository
using BlazorPeliculas.Shared.Entities;
using System;
using System.Collections.Generic;
    
```

```
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class HomePageDTO {
        public List<Movie>? OnBoard { get; set; }
        public List<Movie>? NextReleases { get; set; }

    }
}
```

Hacemos el GET para obtener los dos listados:

```
MoviesController.cs
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context, IFileSaver fileSaver) {
            this.context = context;
            this.fileSaver = fileSaver;
        }

        [HttpGet]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
                .Take(limit)
                .OrderBy(movie => movie.ReleaseDate)
                .ToListAsync();
        }
}
```

```
var result = new HomePageDTO {
    OnBoard = onBoardMovies,
    NextReleases = nextReleases
};

return result;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
    }

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}
}
```

Agregamos el namespace de DTOs en _Imports.cs

```
_Imports.razor
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorPeliculas.Client
@using BlazorPeliculas.Client.Shared
@using BlazorPeliculas.Client.Utilities
@using BlazorPeliculas.Shared.Entities
@using BlazorPeliculas.Client.Repositories
@using BlazorPeliculas.Client.Helpers
@using CurrieTechnologies.Razor.SweetAlert2
@using BlazorPeliculas.Shared.DTOs
```

Vamos a Index.razor y duplicamos el listado (para poder mostrar ambos) y hacemos el GET.

```
Index.razor
@page "/"
@inject IRepository repository
```

```
<PageTitle>Blazor Movies</PageTitle>

<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
        OnBoard = responseHTTP.Response!.OnBoard;
        NextReleases = responseHTTP.Response!.NextReleases;
    }
}
```

Al correr el código, vemos que tenemos una película en cartel y ningún futuro estreno.

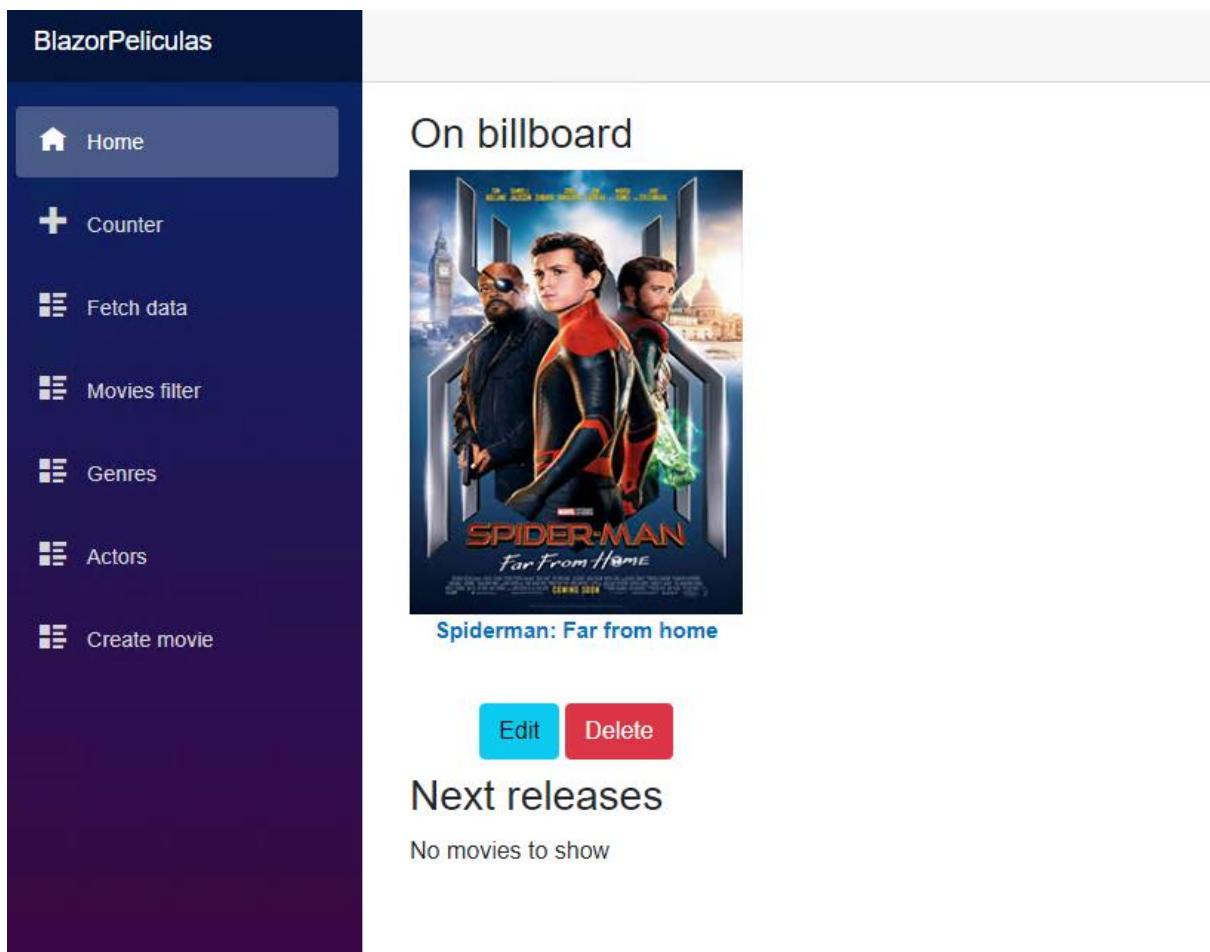
On billboard



Next releases



Y luego de unos segundos...



Vemos que el link en la película no dirige correctamente. Agregamos una función en [Movie.cs](#) para tener el título listo para URL:

```
_Imports.razor
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public int ID { get; set; }
        [Required]
        public string Title { get; set; } = null!;
        public string? Summary { get; set; }
        public bool OnBillboard { get; set; }
        public string? Trailer { get; set; }
        public DateTime? ReleaseDate { get; set; }
        public string? Poster { get; set; }
        public List<GenresMovie> GenresMovie { get; set; } = new List<GenresMovie>();
```

```

public List<MovieActor> MovieActor { get; set; } = new List<MovieActor>();
public string? TrimmedTitle {
    get {
        if(string.IsNullOrEmpty(Title)) {
            return null;
        }

        if(Title.Length > 60) {
            return Title.Substring(0, 60) + "...";
        }
        else {
            return Title;
        }
    }
}

public string? urlTitle() {
    return Title.Replace(" ", "-");
}
}
}
}

```

Y luego corregimos en **MovieItem.razor** para hacer el **NavigateTo** correctamente:

MovieItem.razor

```

<div class="me-2 mb-2" style="text-align:center">
    <a href="@urlMovie">
        
    </a>
    <p style="max-width:225px;height:44px;font-size:15px;font-weight:bold;">
        <a href="@urlMovie" class="text-decoration-none">@Movie.Title</a>
    </p>
</div>
    <a class="btn btn-info">Edit</a>
    <button type="button" class="btn btn-danger"
        @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
</div>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }

    private string urlMovie = string.Empty;
    protected override void OnInitialized() {

```

```
urlMovie = $"./movie/{Movie.ID}/{Movie.urlTitle()}";  
}  
}
```

Visualizar película

Haremos un endpoint en nuestro Controller de películas que recibirá un ID de tipo entero. Como queremos traer datos de los géneros y los actores, aprovecharemos las propiedades de navegación que habíamos agregado. Para ello usaremos el **.Include()** y pedirle la propiedad de navegación **GenresMovie**. También usaremos **.ThenInclude()**. Esto es porque en **GenresMovie** tenemos los IDs de los géneros. Tenemos que esperar a que se haya hecho la primera inclusión para poder incluir a través de la propiedad de navegación de **Genres** de la clase **GenresMovie** la información del **Genre** que queremos.

Hacemos lo mismo (**Include + ThenInclude**) con los actores, con la diferencia de que a este Include le agregamos un **OrderBy** para que estén ordenados según la importancia del personaje en la película.

Luego de los filtros usamos **FirstOrDefaultAsync** porque estamos filtrando por ID y debería ser suficiente para que traiga uno sólo. De encontrar más, traer sólo el primero o el valor por defecto.

Por el momento, no tenemos el sistema de votación por lo que hardcodearemos como si hubieran votos.

En el método **Post** agregamos el **FOR** para guardar el Orden correctamente en el campo correspondiente para que el mismo sea impactado en la base de datos.

Antes de seguir, creamos una clase **pública** llamada **MovieViewDTO** en la carpeta **DTOs** del proyecto **Shared**.

MovieViewDTO.cs

```
using BlazorPeliculas.Shared.Entities;  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace BlazorPeliculas.Shared.DTOs {  
    public class MovieViewDTO {  
        public Movie Movie { get; set; } = null!;  
        public List<Genre> Genres { get; set; } = new List<Genre>();  
        public List<Actor> Actors { get; set; } = new List<Actor>();  
        public int UserVote { get; set; }  
        public double VotesMedia { get; set; }  
    }  
}
```

En **model.Genres** hacemos una proyección desde **GenresMovie**. Esto es porque en **GenresMovie** es del tipo de dato **GenresMovie** pero nosotros queremos los géneros así que tomamos los datos de ahí.

Referencias cíclicas

Otra cosa que vamos a necesitar es configurar para que ese serializador de JSON manejen las referencias cíclicas. ¿qué es eso?

En este **GET** que estamos haciendo retornaremos **MovieViewDTO** en la que tenemos una referencia a **Movie**.

MovieViewDTO.cs

```
public class MovieViewDTO {
    public Movie Movie { get; set; } = null!;
    public List<Genre> Genres { get; set; } = new List<Genre>();
    public List<Actor> Actors { get; set; } = new List<Actor>();
    public int UserVote { get; set; }
    public double VotesMedia { get; set; }
}
```

En **Movie** hay una referencia a **GenresMovie**:

Movie.cs

```
public class Movie {
    public int ID { get; set; }
    [Required]
    public string Title { get; set; } = null!;
    public string? Summary { get; set; }
    public bool OnBillboard { get; set; }
    public string? Trailer { get; set; }
    public DateTime? ReleaseDate { get; set; }
    public string? Poster { get; set; }
    public List<GenresMovie> GenresMovie { get; set; } = new
    List<GenresMovie>();
    public List<MovieActor> MovieActor { get; set; } = new
    List<MovieActor>();
    public string? TrimmedTitle { ... }

    public string? urlTitle() { ... }
}
```

y **GenresMovie** tiene una referencia a la **Movie**:

GenresMovie.cs

```
public class GenresMovie {
    public int MovieID { get; set; }
```

```
public int GenreID { get; set; }
public Genre? Genre { get; set; }
public Movie? Movie { get; set; }
```

O sea, tenemos una relación circular. Es decir:

MovieViewDTO -> Movie -> GenresMovie -> Movie

Es decir, hay una referencia cíclica entre **Movie** y **GenresMovie**. Por defecto, el serializador de JSON arroja un error cuando intentamos retornar datos que tienen referencias cíclicas.

Para configurar esto, agregamos la opción correspondiente en el **Program.cs** del proyecto **Server**.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews()
    .AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler =
    ReferenceHandler.IgnoreCycles);
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
    options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
```

```
app.UseStaticFiles();
```

```
app.UseRouting();
```

```
app.MapRazorPages();
```

```
app.MapControllers();
```

```
app.MapFallbackToFile("index.html");
```

```
app.Run();
```

El código del Controller es:

MoviesController.cs

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;
```

```
namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context, IFileSaver fileSaver) {
            this.context = context;
            this.fileSaver = fileSaver;
        }

        [HttpGet]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
```

```
.Take(limit)
.OrderBy(movie => movie.ReleaseDate)
.ToListAsync();

var result = new HomePageDTO {
    OnBoard = onBoardMovies,
    NextReleases = nextReleases
};

return result;
}

[HttpGet("{id:int}")]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if(movie is null) {
        //No se encontró la película
        return NotFound();
    }

    //TODO: Sistema de votación.
    var votesMedia = 4;
    var userVote = 5;

    var model = new MovieViewDTO();
    model.Movie = movie;
    model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
    model.Actors = movie.MovieActor.Select(ma => new Actor {
        Name = ma.Actor!.Name,
        Photo = ma.Actor.Photo,
        Character = ma.Character,
        ID = ma.Actor.ID
    }).ToList();

    model.VotesMedia= votesMedia;
    model.UserVote= userVote;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
```

```
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
    }

    if(movie.MovieActor is not null) {
        for(int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}
}
```

Como me cansé de repetir la imagen de la ruedita, creamos el **LoadingWheel.razor** en la carpeta **Shared** del proyecto **Client**.

>LoadingWheel.cs

```

```

Y el ViewMovie sería:

ViewMovie.razor

```
@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
@inject IRepository repository
@inject SweetAlertService swal
<h3>ViewMovie</h3>

@if(model is null) {
    <LoadingWheel />
}
else {
    <h2>@movie.Title (@movie.ReleaseDate!.Value.ToString("yyyy"))</h2>
}

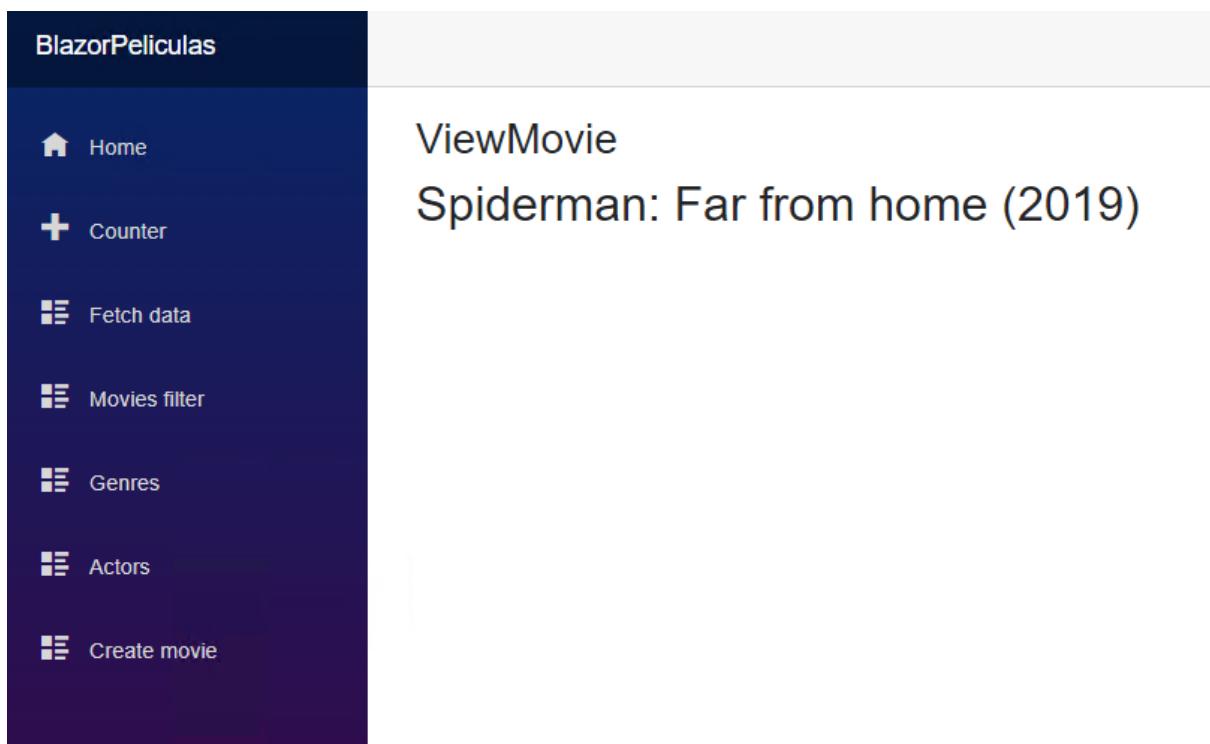
@code {
    [Parameter] public int MovieID { get; set; }
    [Parameter] public string MovieName { get; set; } = null!;
    private MovieViewDTO? model;
    private Movie movie = null!;

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<MovieViewDTO>($"api/movies/{MovieID}");
    }
}
```

```

if (responseHTTP.Error) {
    var ErrMessage = await responseHTTP.GetErrMsg();
    await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
}
else {
    model = responseHTTP.Response;
    movie = model!.Movie;
}
}
    
```

Con estos cambios, la visualización de la película se ve así:



Para poder mostrar los géneros, agregaremos otra película que sí tenga los géneros guardados en la BD.

Agregamos el resto de los cambios. El código queda así.

```

ViewMovie.razor
@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
@inject IRepository repository
@inject SweetAlertService swal

@if(model is null) {
    <LoadingWheel />
}
else {
    <h2>@movie.Title (@movie.ReleaseDate!.Value.ToString("yyyy"))</h2>
}
    
```

```

@foreach(var genre in model.Genres) {
    <a class="me-2 badge bg-primary rounded-pill text-decoration-none"
    href="movies/filter?genred=@genre.ID">@genre.Name</a>
}

<span>| @movie.ReleaseDate!.Value.ToString("dd MM yyyy")
| Media: @model.VotesMedia.ToString("0.#")/5
| Your vote: @model.UserVote.ToString()</span>

<div class="d-flex mt-2">
    <span style="display:inline-block;" class="me-2">
        
    </span>

    <iframe width="560" height="315" src="https://www.youtube.com/embed/@movie.Trailer"
    title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write;
    encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
</div>

<div class="mt-2">
    <h3>Summary</h3>
    <div>
        <ShowMD MDContent="@movie.Summary" />
    </div>
</div>

<div class="mt-2">
    <h3>Actors</h3>
    <div class="d-flex flex-column">
        @foreach(var actor in model.Actors) {
            <div class="mb-2">
                
                <span style="display:inline-block;width:200px;">@actor.Name</span>
                <span style="display:inline-block;width:45px;">...</span>
                <span>@actor.Character</span>
            </div>
        }
    </div>
</div>
}

@code {
    [Parameter] public int MovieID { get; set; }
    [Parameter] public string MovieName { get; set; } = null!;
    private MovieViewDTO? model;
    private Movie movie = null!;

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<MovieViewDTO>($"api/movies/{MovieID}");

        if (responseHTTP.Error) {
    
```

```

var ErrMessage = await responseHTTP.GetErrMessage();
await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
}
else {
    model = responseHTTP.Response;
    movie = model!.Movie;
}
}
}
    
```

Por el momento, se vería así:

Actualizaciones

Géneros

Lo primero que necesitamos es que el botón Editar del índice de géneros tenga acción para ello, agregamos el href.

GenresList.razor

```

@page "/genres"
@inject IRepository repository
<h3>Genres</h3>

<div class="mb-3">
    
```

```

<a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

<GenericList List="Genres">
<HasRecordsComplete>
    <table class="table table-striped">
        <thead>
            <tr>
                <th></th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            @foreach(var item in Genres!) {
                <tr>
                    <td>
                        <a href="/genres/edit/@item.ID" class="btn btn-success">Edit</a>
                        <button class="btn btn-danger">Delete</button>
                    </td>
                    <td>@item.Name</td>
                </tr>
            }
        </tbody>
    </table>
</HasRecordsComplete>
</GenericList>

@code {
    public List<Genre>? Genres { get; set; }

    protected async override Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<Genre>>("api/genres");
        Genres = responseHTTP.Response!;
    }
}

```

A continuación, tenemos que traer la información actual del género. Para ello, inyectamos el repositorio y hacemos el get correspondiente. Para conseguir esto el evento a usar es **OnInitializedAsync** y debe ser una **async Task** (como en todos los anteriores).

Al igual que hicimos antes, no queremos mostrar la info hasta que se haya traído la misma desde la BD.

EditGenre.razor

```

@page "/genres/edit/{GenreID:int}"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal

```

```

<h3>EditGenre</h3>
@if(Genre is not null) {
    <GenreForm @ref="genreForm" Genre="Genre" OnValidSubmit="Edit" />
}
else {
    <LoadingWheel/>
}

@code {
    [Parameter] public int GenreID { get; set; }
    private Genre? Genre;
    private GenreForm? genreForm;

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<Genre>($"api/genres/{GenreID}");

        if(responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("genres");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            Genre = responseHTTP.Response;
        }
    }

    private void Edit() {
        Console.WriteLine("Ejecuting Edit method");
        Console.WriteLine($"Genre ID: {Genre!.ID}");
        Console.WriteLine($"Genre Name: {Genre!.Name}");

        genreForm!.formPostedCorrectly = true;
        navManager.NavigateTo("/genres");
    }
}

```

Y agregamos el endpoint para el GET(id:int) en la controller de géneros como también el PUT para poder actualizar los cambios:

GenresController.razor

```

using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;

```

```
namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Genre>>> Get() {
            return await context.Genres.ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Genre>> Get(int id) {
            var genre = await context.Genres.FirstOrDefaultAsync(genre => genre.ID == id);

            if (genre is null)
                return NotFound();

            return genre;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            context.Add(genre);
            await context.SaveChangesAsync();
            return genre.ID;
        }

        [HttpPut]
        public async Task<ActionResult<int>> Put(Genre genre) {
            context.Update(genre); //Marco el género para ser actualizado.
            await context.SaveChangesAsync(); //Se hace el UPDATE
            return NoContent(); //Todo se hizo correctamente
        }
    }
}
```

Copiamos el método **Post** de Repository para hacer el **Put**. Luego, hacemos el Pull para que la firma se incluya en la interfaz.

Repository.cs

```
using BlazorPeliculas.Shared.Entities;
using System.Text;
using System.Text.Json;

namespace BlazorPeliculas.Client.Repositories {
```

```
public class Repository : IRepository {
    public HttpClient HttpClient { get; }

    public Repository(HttpClient httpClient) {
        HttpClient = httpClient;
    }
    private JsonSerializerOptions DefaultJsonOptions = new JsonSerializerOptions {
        PropertyNameCaseInsensitive = true
    };

    //Lo haremos con T porque podríamos recibir un listado de géneros, actores, películas, etc.
    public async Task<HttpResponseWrapper<T>> Get<T>(string url) {
        var responseHTTP = await HttpClient.GetAsync(url);
        if (responseHTTP.IsSuccessStatusCode) {
            var response = await DeserializeResponse<T>(responseHTTP, DefaultJsonOptions);
            return new HttpResponseWrapper<T>(response, Error: false, responseHTTP);
        }
        else {
            return new HttpResponseWrapper<T>(default, Error:true, responseHTTP);
        }
    }

    public async Task<HttpResponseWrapper<object>> Post<T>(string url, T send) {
        var sendJSON = JsonSerializer.Serialize(send);
        var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
        var responseHttp = await HttpClient.PostAsync(url, sendContent);
        return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
    }

    public async Task<HttpResponseWrapper<object>> Put<T>(string url, T send) {
        var sendJSON = JsonSerializer.Serialize(send);
        var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
        var responseHttp = await HttpClient.PutAsync(url, sendContent);
        return new HttpResponseWrapper<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
    }

    public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send) {
        var sendJSON = JsonSerializer.Serialize(send);
        var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
        var responseHttp = await HttpClient.PostAsync(url, sendContent);

        if(responseHttp.IsSuccessStatusCode) {
            var response = await DeserializeResponse<TResponse>(responseHttp, DefaultJsonOptions);
            return new HttpResponseWrapper<TResponse>(response, false, responseHttp);
        }
        return new HttpResponseWrapper<TResponse>(default, !responseHttp.IsSuccessStatusCode,
        responseHttp);
    }

    private async Task<T> DeserializeResponse<T>(HttpResponseMessage httpResponse,
```

```
JsonSerializerOptions jsonSerializerOptions) {
    var responseString = await httpResponse.Content.ReadAsStringAsync();
    return JsonSerializer.Deserialize<T>(responseString, jsonSerializerOptions);
}

public List<Movie> GetMovies() {
    return new List<Movie>() {
        new Movie {Title = "Wakanda forever",
            ReleaseDate = new DateTime(2022, 11, 11),
            Poster =
"https://upload.wikimedia.org/wikipedia/en/thumb/3/3b/Black_Panther_Wakanda_Forever_poster.jpg/220px-
x-Black_Panther_Wakanda_Forever_poster.jpg"},
        new Movie {Title = "Moana",
            ReleaseDate = new DateTime(2016, 11, 23),
            Poster =
"https://upload.wikimedia.org/wikipedia/en/thumb/2/26/Moana_Teaser_Poster.jpg/220px-
Moana_Teaser_Poster.jpg" },
        new Movie {Title = "Inception",
            ReleaseDate = new DateTime(2010, 7, 16),
            Poster =
"https://upload.wikimedia.org/wikipedia/en/2/2e/Inception_%282010%29_theatrical_poster.jpg" }
    };
}
}
```

IRepository.cs

```
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public interface IRepository {
        Task<HttpResponseWrapper<T>> Get<T>(string url);
        List<Movie> GetMovies();
        Task<HttpResponseWrapper<object>> Post<T>(string url, T send);
        Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send);
        Task<HttpResponseWrapper<object>> Put<T>(string url, T send);
    }
}
```

EditGenre.razor

```
@page "/genres/edit/{GenreID:int}"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal

<h3>Edit genre</h3>
@if(Genre is not null) {
    <GenreForm @ref="genreForm" Genre="Genre" OnValidSubmit="Edit" />
```

```
}

else {
    <LoadingWheel/>
}

@code {
    [Parameter] public int GenreID { get; set; }
    private Genre? Genre;
    private GenreForm? genreForm;

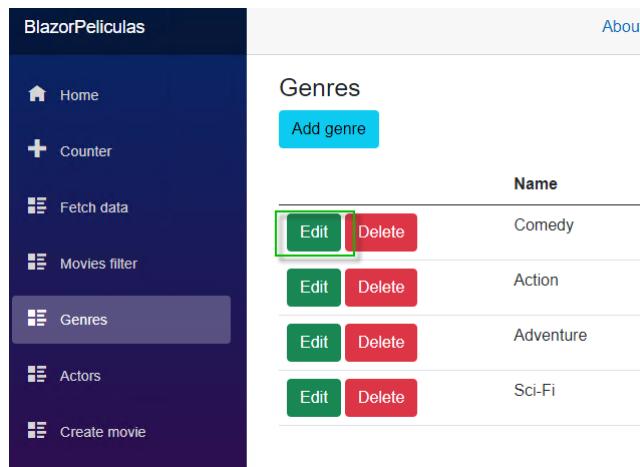
    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<Genre>($"api/genres/{GenreID}");

        if(responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("genres");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            Genre = responseHTTP.Response;
        }
    }

    private async Task Edit() {
        var responseHTTP = await repository.Put("/api/genres", Genre);

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            genreForm!.formPostedCorrectly = true;
            navManager.NavigateTo("/genres");
        }
    }
}
```

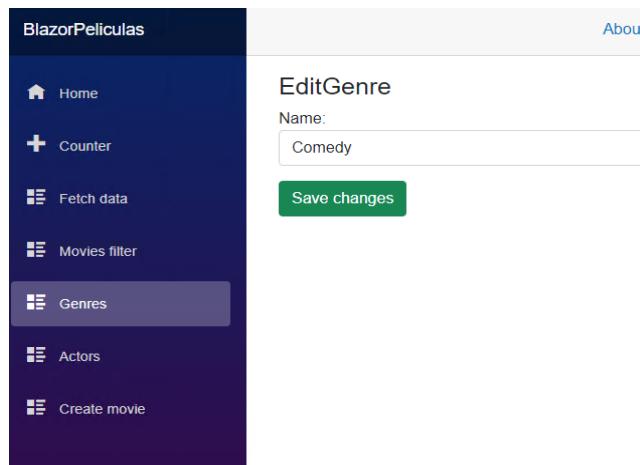
Ejecutamos y vamos a editar el primer género:



The screenshot shows the Blazor application's navigation menu on the left with "Genres" selected. The main content area displays a table titled "Genres" with four rows. The first row has "Name" as "Comedy", with "Edit" and "Delete" buttons. The other three rows have names "Action", "Adventure", and "Sci-Fi" respectively, each with "Edit" and "Delete" buttons. The "Edit" button for the first row is highlighted with a green border.

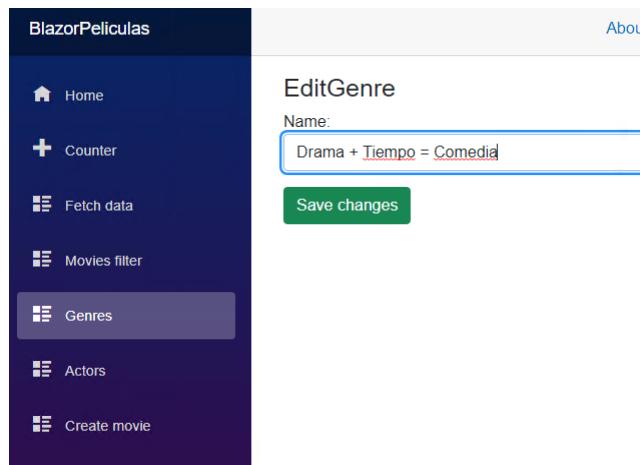
Name
Comedy
Action
Adventure
Sci-Fi

Nos muestra el nombre original para que lo cambiemos:



The screenshot shows the Blazor application's navigation menu on the left with "Genres" selected. The main content area displays a form titled "EditGenre". It has a "Name:" label and a text input field containing "Comedy". Below the input field is a "Save changes" button.

Cambiamos el mismo:



The screenshot shows the Blazor application's navigation menu on the left with "Genres" selected. The main content area displays the same "EditGenre" form as before, but the text input field now contains "Drama + Tiempo = Comedia". The "Save changes" button is visible below the input field.

Al presionar el botón de grabar, se va al índice con el nombre cambiado:

Name		
Drama + Tiempo = Comedia	Edit	Delete
Action	Edit	Delete
Adventure	Edit	Delete
Sci-Fi	Edit	Delete

Actores

Hacemos cambios muy similares a los de recién para EditActor.razor y ActorsList.razor.

EditActor.razor

```
@page "/actors/edit/{ActorID:int}"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
<h3>Edit actor</h3>

@if(Actor is not null) {
    <ActorsForm Actor="Actor" OnValidSubmit="Edit" />
}
else {
    <LoadingWheel/>
}

@code {
    [Parameter] public int ActorID { get; set; }
    Actor? Actor;

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<Actor>($"api/actors/{ActorID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("actors");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
    }
}
```

```
else {
    Actor = responseHTTP.Response;
}
}

private async Task Edit() {
    var responseHTTP = await repository.Put("/api/actors", Actor);

    if (responseHTTP.Error) {
        var ErrMessage = await responseHTTP.GetErrMsg();
        await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
    }
    else {
        navManager.NavigateTo("/actors");
    }
}
```

ActorsList.razor

```
@page "/actors"
@inject IRepository repository
<h3>Actors list</h3>

<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>

<GenericList List="Actors">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Actors!) {
                    <tr>
                        <td>
                            <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                            <button class="btn btn-danger">Delete</button>
                        </td>
                        <td>@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
```

```
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }

    protected async override Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<Actor>>("api/actors");
        Actors = responseHTTP.Response!;
    }
}
```

El **Get** es muy similar al de géneros. El **Put**, es un poco distinto porque tenemos el campo **Photo** que la actualizaremos manualmente. Esto es porque si el campo viene nulo significa que el usuario no modificó la foto.

AutoMapper

Para eso, utilizaremos **AutoMapper** que es una librería que nos permite mapear de un objeto a otro. Instalamos el NuGet:

[AutoMapper.Extensions.Microsoft.DependencyInjection](#)

Luego agregamos el servicio en el **Program.cs** del proyecto **Server**.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews()
    .AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler =
    ReferenceHandler.IgnoreCycles);
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
    options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
builder.Services.AddAutoMapper(typeof(Program));

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
```

```
app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

Para conseguir esto, creamos el archivo **AutoMapperProfile.cs** en la carpeta **Helpers** del proyecto **Server** a cuya clase la hacemos heredar de **Profile** en cuyo constructor creamos un mapeo de actor a actor y le indicamos que el campo **Photo** debe ser ignorado.

```
AutoMapperProfile.cs
using AutoMapper;
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Server.Helpers {
    public class AutoMapperProfile : Profile {
        public AutoMapperProfile() {
            CreateMap<Actor, Actor>()
                .ForMember(x => x.Photo, option => option.Ignore());
        }
    }
}
```

Y la controladora, entonces, queda así:

```
ActorsController.cs
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.Entities;
```

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "people";

        public ActorsController(ApplicationDbContext context, IFileSaver fileSaver, IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Actor>>> Get() {
            return await context.Actors.ToListAsync();
        }

        [HttpGet("search/{searchText}")]
        public async Task<ActionResult<IEnumerable<Actor>>> Get(string searchText) {
            if(string.IsNullOrWhiteSpace(searchText))
                return new List<Actor>();

            searchText = searchText.ToLower();
            return await context.Actors
                .Where(x => x.Name.ToLower().Contains(searchText))
                .Take(5)
                .ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Actor>> Get(int id) {
            var actor = await context.Actors.FirstOrDefaultAsync(actor => actor.ID == id);

            if(actor is null)
                return NotFound();

            return actor;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Actor actor) {
            if(!string.IsNullOrWhiteSpace(actor.Photo)) {
                //Nos mandaron una foto desde el frontend
                var photoActor = Convert.FromBase64String(actor.Photo);
                actor.Photo = await fileSaver.SaveFile(photoActor, ".jpg", container);
            }
        }
    }
}
```

```
        }

        context.Add(actor);
        await context.SaveChangesAsync();
        return actor.ID;
    }

    [HttpPost]
    public async Task<ActionResult> Put(Actor actor) {
        var actorDB = await context.Actors.FirstOrDefaultAsync(a => a.ID == actor.ID);

        if(actorDB is null)
            return NotFound();
        //Tomá las propiedades de actor y pasalas a actorDB
        actorDB = mapper.Map(actor, actorDB);

        if(!string.IsNullOrWhiteSpace(actor.Photo)) {
            //Nos mandaron una foto desde el frontend
            var photoActor = Convert.FromBase64String(actor.Photo);
            actorDB.Photo = await fileSaver.EditFile(photoActor, ".jpg", container, actorDB.Photo);
        }

        await context.SaveChangesAsync(); //Se hace el UPDATE
        return NoContent();           //Todo se hizo correctamente
    }
}
```

Probemos. Editaremos a Tom Holland.

Actors list

Create actor

Name	
Edit Delete	Tom Holland
Edit Delete	Tom Hanks
Edit Delete	Leonardo DiCaprio
Edit Delete	Joseph Gordon-Levitt

Nos trae los datos y le agregamos un 2 al nombre:

Edit actor

Name:

Birthdate:

Photo

 No file chosen

Al grabar, se va al índice y lo vemos actualizado. Si volviéramos a entrar veríamos que la foto no se perdió.

Actors list

		Name
<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	Tom Holland 2
<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	Tom Hanks
<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	Leonardo DiCaprio
<input type="button" value="Edit"/>	<input type="button" value="Delete"/>	Joseph Gordon-Levitt

Sólo para que quede más lindo, agregaremos la foto en el listado de actores:

ActorsList.razor

```
@page "/actors"
@inject IRepository repository
<h3>Actors list</h3>

<div class="mb-3">
```

```
<a href="actors/create" class="btn btn-info">Create actor</a>
</div>

<GenericList List="Actors">
<HasRecordsComplete>
    <table class="table table-striped">
        <thead>
            <tr>
                <th></th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Actors!) {
                <tr>
                    <td>
                        <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                        <button class="btn btn-danger">Delete</button>
                    </td>
                    <td>&nbsp;@item.Name</td>
                </tr>
            }
        </tbody>
    </table>
</HasRecordsComplete>
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }

    protected async override Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<Actor>>("api/actors");
        Actors = responseHTTP.Response!;
    }
}
```

Actors list

[Create actor](#)

	Name
Edit	Delete
	Tom Holland
Edit	Delete
	Tom Hanks
Edit	Delete
	Leonardo DiCaprio
Edit	Delete
	Joseph Gordon-Levitt

Películas

Lo primero es agregar el URL para editar la película. Cambiamos la variable existente y agregamos el href correspondiente al anchor de **Edit**:

MovieItem.razor

```

<div class="me-2 mb-2" style="text-align:center">
    <a href="@urlViewMovie">
        
    </a>
    <p style="max-width:225px;height:44px;font-size:15px;font-weight:bold;">
        <a href="@urlViewMovie" class="text-decoration-none">@Movie.Title</a>
    </p>
    <div>
        <a class="btn btn-info" href="@urlEditMovie">Edit</a>
        <button type="button" class="btn btn-danger"
            @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
    </div>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }

    private string urlViewMovie = string.Empty;
    private string urlEditMovie = string.Empty;
    protected override void OnInitialized()
    {
        urlViewMovie = $"/movie/{Movie.ID}/{Movie.urlTitle()}";
        urlEditMovie = $"/movie/edit/{Movie.ID}";
    }
}

```

Abrimos el componente EditMovie.razor y vemos que necesitaremos la película, los géneros seleccionados, los no seleccionados y el listado de actores.

El GET que hicimos anteriormente no está completo. Porque sólo trae los listados de géneros de la película, pero no los géneros no seleccionados. Por tal motivo, reutilizaremos el método, pero crearemos uno nuevo **PutGet** (que es como que el **Put** de actualizar, pero es el **Get** para poder cargar la info. Crearemos un nuevo DTO en la carpeta **DTOs** del proyecto **Shared** llamado **MovieUpdateDTO.cs**.

MovieUpdateDTO.cs

```
using BlazorPeliculas.Shared.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class MovieUpdateDTO {
        public Movie Movie { get; set; } = null!;
        public List<Actor> Actors { get; set; } = new List<Actor>();
        public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
        public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();
    }
}
```

Ya tenemos todo para hacer el PutGet:

MoviesController.cs

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context, IFileSaver fileSaver) {
            this.context = context;
            this.fileSaver = fileSaver;
        }

        [HttpGet]
```

```
public async Task<ActionResult<HomePageDTO>> Get() {
    var limit = 6;
    var onBoardMovies = await context.Movies
        .Where(movie => movie.OnBillboard)
        .Take(limit)
        .OrderByDescending(movie => movie.ReleaseDate)
        .ToListAsync();
    var today = DateTime.Today;
    var nextReleases = await context.Movies
        .Where(movie => movie.ReleaseDate > today)
        .Take(limit)
        .OrderBy(movie => movie.ReleaseDate)
        .ToListAsync();

    var result = new HomePageDTO {
        OnBoard = onBoardMovies,
        NextReleases = nextReleases
    };
    return result;
}

[HttpGet("{id:int}")]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if(movie is null) {
        //No se encontró la película
        return NotFound();
    }

    //TODO: Sistema de votación.
    var votesMedia = 4;
    var userVote = 5;

    var model = new MovieViewDTO();
    model.Movie = movie;
    model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
    model.Actors = movie.MovieActor.Select(ma => new Actor {
        Name = ma.Actor!.Name,
        Photo = ma.Actor.Photo,
        Character = ma.Character,
        ID = ma.Actor.ID
    }).ToList();
}
```

```
model.VotesMedia= votesMedia;
model.UserVote= userVote;

    return model;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if(movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
    }

    if(movie.MovieActor is not null) {
        for(int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}
}
```

Y en EditMovie.cs hacemos el PutGet correspondiente y mostramos la info en pantalla:

EditMovie.razor

```
@page "/movie/edit/{MovieID:int}"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
<h3>Edit Movie</h3>

@if(Movie is not null) {
<MoviesForm Movie="Movie" OnValidSubmit="Edit"
    UnselectedGenres="Unselected"
    SelectedGenres="Selected"
    SelectedActors="SelectedActors"/>
}
else {
    <LoadingWheel />
}

@code {
    [Parameter] public int MovieID { get; set; }
    private Movie? Movie;
    private List<Genre> Unselected = new List<Genre>();
    private List<Genre> Selected = new List<Genre>();
    private List<Actor> SelectedActors = new List<Actor>();

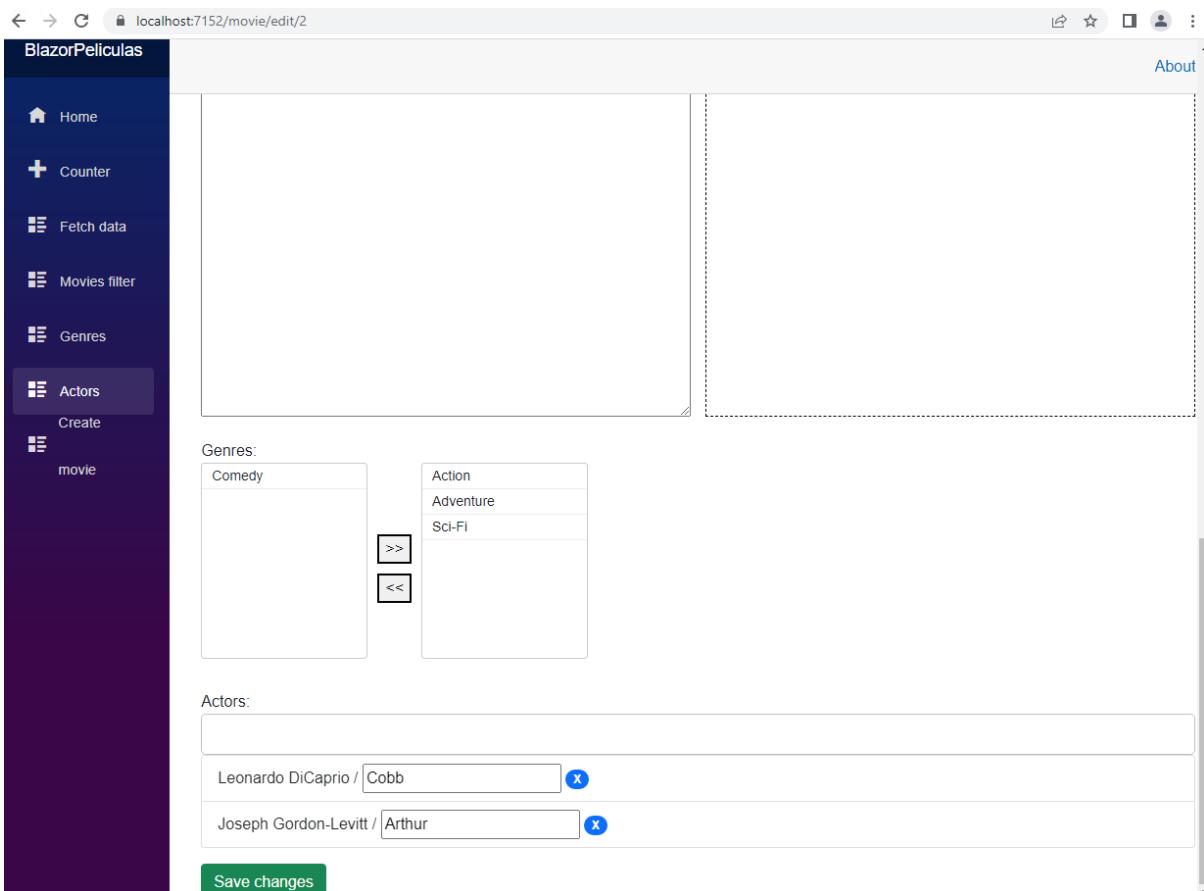
    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.PutGet<MovieUpdateDTO>($"api/movies/update/{MovieID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            MovieUpdateDTO model = responseHTTP.Response!;

            Movie = model.Movie;
            Unselected = model.UnselectedGenres;
            Selected = model.SelectedGenres;
            SelectedActors = model.Actors;
        }
    }

    private void Edit() {
        Console.WriteLine("Editing movie...");
    }
}
```

Al probar, vemos que trae los géneros seleccionados, los no seleccionados y los actores:



Por el momento, sólo trajimos la info de la película. Faltaría el PUT correspondiente para actualizar la información que se haya modificado. Lo primero que tenemos que hacer es ignorar el campo Poster para que el AutoMapper no lo mapee (ni lo actualice automáticamente).

AutoMapperProfile.cs

```
using AutoMapper;
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Server.Helpers {
    public class AutoMapperProfile: Profile {
        public AutoMapperProfile() {
            CreateMap<Actor, Actor>()
                .ForMember(x => x.Photo, option => option.Ignore());

            CreateMap<Movie, Movie>()
                .ForMember(x => x.Poster, option => option.Ignore());
        }
    }
}
```

Al momento de traer la info en el método PUT desde context.Movies incluimos la información de **GenresMovie** y **MovieActor** para que al hacer el automap se actualice toda la info automáticamente.

También hay que tener en cuenta que el usuario puede cambiar el orden de los actores. Por lo que habría que modificar esa información también. Para conseguir ello, marcamos el código que escribimos antes y lo extraemos a un método nuevo.

EditMovie.razor

```
@page "/movie/edit/{MovieID:int}"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal
<h3>Edit Movie</h3>

@if(Movie is not null) {
<MoviesForm Movie="Movie" OnValidSubmit="Edit"
    UnselectedGenres="Unselected"
    SelectedGenres="Selected"
    SelectedActors="SelectedActors"/>
}
else {
    <LoadingWheel />
}

@code {
    [Parameter] public int MovieID { get; set; }
    private Movie? Movie;
    private List<Genre> Unselected = new List<Genre>();
    private List<Genre> Selected = new List<Genre>();
    private List<Actor> SelectedActors = new List<Actor>();

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.PutGet<MovieUpdateDTO>($"api/movies/update/{MovieID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            MovieUpdateDTO model = responseHTTP.Response!;

            Movie = model.Movie;
            Unselected = model.UnselectedGenres;
            Selected = model.SelectedGenres;
            SelectedActors = model.Actors;
        }
    }
}
```

```
    }

}

private async Task Edit() {
    var responseHTTP = await repository.Put("/api/movies", Movie);

    if (responseHTTP.Error) {
        var ErrMessage = await responseHTTP.GetErrMsg();
        await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
    }
    else {
        navManager.NavigateTo($"/movie/{MovieID}");
    }
}
```

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
        }
    }
}
```

```
var today = DateTime.Today;
var nextReleases = await context.Movies
    .Where(movie => movie.ReleaseDate > today)
    .Take(limit)
    .OrderBy(movie => movie.ReleaseDate)
    .ToListAsync();

var result = new HomePageDTO {
    OnBoard = onBoardMovies,
    NextReleases = nextReleases
};
return result;
}

[HttpGet("{id:int}")]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }

    //TODO: Sistema de votación.
    var votesMedia = 4;
    var userVote = 5;

    var model = new MovieViewDTO();
    model.Movie = movie;
    model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
    model.Actors = movie.MovieActor.Select(ma => new Actor {
        Name = ma.Actor!.Name,
        Photo = ma.Actor.Photo,
        Character = ma.Character,
        ID = ma.Actor.ID
    }).ToList();

    model.VotesMedia = votesMedia;
    model.UserVote = userVote;

    return model;
}

[HttpGet("edit/{id:int}")]
```

```
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPut]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);
```

```
if (movieDB == null)
    return NotFound();

//Tomá las propiedades de movie y pasalas a movieDB
movieDB = mapper.Map(movie, movieDB);

if (!string.IsNullOrWhiteSpace(movie.Poster)) {
    //Nos mandaron una foto desde el frontend
    var Poster = Convert.FromBase64String(movie.Poster);
    movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
}

WriteActorsOrder(movieDB);

await context.SaveChangesAsync(); //Se hace el UPDATE
return NoContent();           //Todo se hizo correctamente
}
}
}
```

Borrando registros

Creamos el método DELETE en nuestro **Repository.cs** para poder ser utilizado desde las Controllers.

Repository.cs

```
using BlazorPeliculas.Shared.Entities;
using System.Text;
using System.Text.Json;

namespace BlazorPeliculas.Client.Repositories {
    public class Repository : IRepository {
        public HttpClient HttpClient { get; }

        public Repository(HttpClient httpClient) {
            HttpClient = httpClient;
        }
        private JsonSerializerOptions DefaultJsonOptions = new JsonSerializerOptions {
            PropertyNameCaseInsensitive = true
        };

        //Lo haremos con T porque podríamos recibir un listado de géneros, actores, películas, etc.
        public async Task<HttpResponseWrapper<T>> Get<T>(string url) {
            var responseHTTP = await HttpClient.GetAsync(url);
            if (responseHTTP.IsSuccessStatusCode) {
                var response = await DeserializeResponse<T>(responseHTTP, DefaultJsonOptions);
                return new HttpResponseWrapper<T>(response, Error: false, responseHTTP);
            }
        }
    }
}
```

```
else {
    return new HttpResponseMessage<T>(default, Error:true, responseHTTP);
}

public async Task<HttpResponseWrapper<object>> Post<T>(string url, T send) {
    var sendJSON = JsonSerializer.Serialize(send);
    var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
    var responseHttp = await HttpClient.PostAsync(url, sendContent);
    return new HttpResponseMessage<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
}

public async Task<HttpResponseWrapper<object>> Put<T>(string url, T send) {
    var sendJSON = JsonSerializer.Serialize(send);
    var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
    var responseHttp = await HttpClient.PutAsync(url, sendContent);
    return new HttpResponseMessage<object>(null, !responseHttp.IsSuccessStatusCode, responseHttp);
}

public async Task<HttpResponseWrapper<object>> Delete(string url) {
    var responseHTTP = await HttpClient.DeleteAsync(url);

    return new HttpResponseMessage<object>(null, !responseHTTP.IsSuccessStatusCode,
responseHTTP);
}

public async Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send) {
    var sendJSON = JsonSerializer.Serialize(send);
    var sendContent = new StringContent(sendJSON, Encoding.UTF8, "application/json");
    var responseHttp = await HttpClient.PostAsync(url, sendContent);

    if(responseHttp.IsSuccessStatusCode) {
        var response = await DeserializeResponse<TResponse>(responseHttp, DefaultJsonOptions);
        return new HttpResponseMessage<TResponse>(response, false, responseHttp);
    }
    return new HttpResponseMessage<TResponse>(default, !responseHttp.IsSuccessStatusCode,
responseHttp);
}

private async Task<T> DeserializeResponse<T>(HttpResponseMessage httpResponse,
JsonSerializerOptions jsonSerializerOptions) {
    var responseString = await httpResponse.Content.ReadAsStringAsync();
    return JsonSerializer.Deserialize<T>(responseString, jsonSerializerOptions);
}

public List<Movie> GetMovies() {
    return new List<Movie>() {
        new Movie {Title = "Wakanda forever",
        ReleaseDate = new DateTime(2022, 11, 11),
        Poster =
```

```
"https://upload.wikimedia.org/wikipedia/en/thumb/3/3b/Black_Panther_Wakanda_Forever_poster.jpg/220px-  
x-Black_Panther_Wakanda_Forever_poster.jpg"},  
    new Movie {Title = "Moana",  
        ReleaseDate = new DateTime(2016, 11, 23),  
        Poster =  
"https://upload.wikimedia.org/wikipedia/en/thumb/2/26/Moana_Teaser_Poster.jpg/220px-  
Moana_Teaser_Poster.jpg" },  
    new Movie {Title = "Inception",  
        ReleaseDate = new DateTime(2010, 7, 16),  
        Poster =  
"https://upload.wikimedia.org/wikipedia/en/2/2e/Inception_%282010%29_theatrical_poster.jpg" }  
};  
}  
}  
}
```

Hacemos el Pull para llevarlo al [IRepository.cs](#):

```
IRepository.cs
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Client.Repositories {
    public interface IRepository {
        Task<HttpResponseWrapper<object>> Delete(string url);
        Task<HttpResponseWrapper<T>> Get<T>(string url);
        List<Movie> GetMovies();
        Task<HttpResponseWrapper<object>> Post<T>(string url, T send);
        Task<HttpResponseWrapper<TResponse>> Post<T, TResponse>(string url, T send);
        Task<HttpResponseWrapper<object>> Put<T>(string url, T send);
    }
}
```

Géneros

En el componente **GenresList.razor** hacemos el llamado al **DELETE** y si se realiza correctamente llamamos al nuevo método Load (que refactorizamos con el código que estaba en el **OnInitializedAsync**.

```
GenresList.razor
@page "/genres"
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl

<h3>Genres</h3>

<div class="mb-3">
    <a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

<GenericList List="Genres">
```

```

<HasRecordsComplete>
    <table class="table table-striped">
        <thead>
            <tr>
                <th></th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            @foreach(var item in Genres!) {
                <tr>
                    <td>
                        <a href="/genres/edit/@item.ID" class="btn btn-success">Edit</a>
                        <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                    </td>
                    <td>@item.Name</td>
                </tr>
            }
        </tbody>
    </table>
</HasRecordsComplete>
</GenericList>

@code {
    public List<Genre>? Genres { get; set; }

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task Load() {
        var responseHTTP = await repository.Get<List<Genre>>("api/genres");
        Genres = responseHTTP.Response!;
    }

    public async Task Delete(Genre genre) {
        var responseHTTP = await repository.Delete($"api/genres/{genre.ID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            await Load();
        }
    }
}

```

En **GenresController.cs** buscamos los géneros que concuerden con el ID enviado y lo borramos. Si las filas afectadas es distinto de 0, se pudo borrar el género.

GenresController.cs

```
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Genre>>> Get() {
            return await context.Genres.ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Genre>> Get(int id) {
            var genre = await context.Genres.FirstOrDefaultAsync(genre => genre.ID == id);

            if (genre is null)
                return NotFound();

            return genre;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            context.Add(genre);
            await context.SaveChangesAsync();
            return genre.ID;
        }

        [HttpPut]
        public async Task<ActionResult<int>> Put(Genre genre) {
            context.Update(genre); //Marco el género para ser actualizado.
            await context.SaveChangesAsync(); //Se hace el UPDATE
            return NoContent(); //Todo se hizo correctamente
        }

        [HttpDelete("{id:int}")]
        public async Task<ActionResult<int>> Delete(int id) {
```

```

var affectedFiles = await context.Genres
    .Where(x => x.ID == id)
    .ExecuteDeleteAsync();

if(affectedFiles == 0)
    return NotFound();

return NoContent();
}
}
}

```

Vamos al listado de géneros y vemos que el 5 es Prueba.

Genres

[Add genre](#)

			Name
	Edit	Delete	
			Comedy
			Action
			Adventure
			Sci-Fi
			Prueba

dbo.Genres [Data]			GenresController.cs
	ID	Name	
▶	1	Comedy	
	2	Action	
	3	Adventure	
	4	Sci-Fi	
*	5	Prueba	
	NULL	NULL	

Al presionar Delete, el listado se actualiza y en la tabla no existe más el ID=5:

Genres

[Add genre](#)

			Name
	Edit	Delete	
			Comedy
			Action
			Adventure
			Sci-Fi

dbo.Genres [Data] ➔ GenresController.cs		
	ID	Name
▶	1	Comedy
	2	Action
	3	Adventure
	4	Sci-Fi
*	NULL	NULL

Actores

El índice de actores es prácticamente igual al de los géneros.

ActorsList.razor

```
@page "/actors"
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swal
<h3>Actors list</h3>

<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>

<GenericList List="Actors">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>1</td>
                    <td>Tom Hanks</td>
                </tr>
                <tr>
                    <td>2</td>
                    <td>Meryl Streep</td>
                </tr>
                <tr>
                    <td>3</td>
                    <td>Brad Pitt</td>
                </tr>
                <tr>
                    <td>4</td>
                    <td>Angelina Jolie</td>
                </tr>
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>
```

```

</thead>
<tbody>
    @foreach (var item in Actors!) {
        <tr>
            <td>
                <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
            </td>
            <td>&nbsp;@item.Name</td>
        </tr>
    }
</tbody>
</table>
</HasRecordsComplete>
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task Load() {
        var responseHTTP = await repository.Get<List<Actor>>("api/actors");
        Actors = responseHTTP.Response!;
    }

    public async Task Delete(Actor actor) {
        var responseHTTP = await repository.Delete($"api/actors/{actor.ID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            await Load();
        }
    }
}

```

El código de [ActorsController.cs](#) podría ser idéntico al de géneros si no fuera porque yo quiero borrar la imagen de la foto del actor eliminado.

ActorsController.cs

[using AutoMapper;](#)

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "people";

        public ActorsController(ApplicationDbContext context, IFileSaver fileSaver, IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Actor>>> Get() {
            return await context.Actors.ToListAsync();
        }

        [HttpGet("search/{searchText}")]
        public async Task<ActionResult<IEnumerable<Actor>>> Get(string searchText) {
            if(string.IsNullOrWhiteSpace(searchText))
                return new List<Actor>();

            searchText = searchText.ToLower();
            return await context.Actors
                .Where(x => x.Name.ToLower().Contains(searchText))
                .Take(5)
                .ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Actor>> Get(int id) {
            var actor = await context.Actors.FirstOrDefaultAsync(actor => actor.ID == id);

            if(actor is null)
                return NotFound();

            return actor;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Actor actor) {
            if(!string.IsNullOrWhiteSpace(actor.Photo)) {
                //Nos mandaron una foto desde el frontend
            }
        }
    }
}
```

```

        var photoActor = Convert.FromBase64String(actor.Photo);
        actor.Photo = await fileSaver.SaveFile(photoActor, "jpg", container);
    }

    context.Add(actor);
    await context.SaveChangesAsync();
    return actor.ID;
}

[HttpPost]
public async Task<ActionResult> Put(Actor actor) {
    var actorDB = await context.Actors.FirstOrDefaultAsync(a => a.ID == actor.ID);

    if(actorDB is null)
        return NotFound();
    //Tomá las propiedades de actor y pasalas a actorDB
    actorDB = mapper.Map(actor, actorDB);

    if(!string.IsNullOrWhiteSpace(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actorDB.Photo = await fileSaver.EditFile(photoActor, "jpg", container, actorDB.Photo!);
    }

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var actor = await context.Actors.FirstOrDefaultAsync(x => x.ID == id);

    if(actor is null)
        return NotFound();

    context.Remove(actor); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if(!string.IsNullOrWhiteSpace(actor.Photo))
        await fileSaver.DeleteFile(actor.Photo!, container);

    return NoContent();
}
}

```

Vemos en el listado que está Pepe con una foto de Joseph Gordon-Levitt cuya foto está en la carpeta **wwwroot/people** del proyecto **Server**.

Actors list

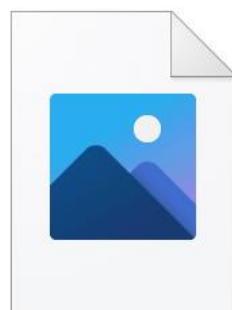
[Create actor](#)

		Name
Edit	Delete	
Edit	Delete	 Tom Holland
Edit	Delete	 Tom Hanks
Edit	Delete	 Leonardo DiCaprio
Edit	Delete	 Joseph Gordon-Levitt
Edit	Delete	 Pepe

repos > BlazorPeliculas > Server > wwwroot > people

[!\[\]\(fea10d9bbf671e524c105b242c5b9338_img.jpg\)](#) [!\[\]\(03b3d9ede2c3e667cb3f28e66b70db97_img.jpg\)](#) [Search people](#)

c994f63-e7e8-4e45-9619-06121b1bb141.jpg



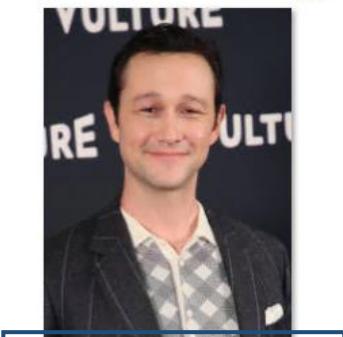
2b89d596-5f00-4ab3-9890-f2e853fbebe96.jpg



26add85f-3f01-4a66-968a-61add2b1f4ad.jpg



340c01c-45a7-409a-ae40-3859c7aef6d5.jpg



b0393281-321b-4217-9573-4fcfa0fe09695.jpg



bd3e3c87-7dc3-496a-ab6e-d1dc66424d3e.jpg

Al presionar el botón de borrar, elimina el actor de la lista (actualiza el listado en pantalla) y borra el archivo de la carpeta mencionada:

Actors list

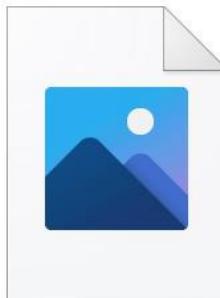
[Create actor](#)

		Name
Edit	Delete	
Edit	Delete	 Tom Holland
Edit	Delete	 Tom Hanks
Edit	Delete	 Leonardo DiCaprio
Edit	Delete	 Joseph Gordon-Levitt

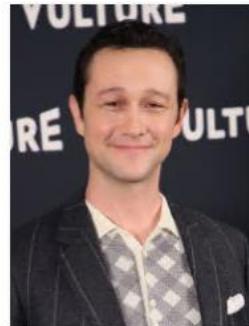
repos > BlazorPeliculas > Server > wwwroot > people

[▼](#)

0c994f63-e7e8-4e45-9619-06121b1bb141.jpg



2b89d596-5f00-4ab3-9890-f2e853fbe96.jpg



26add85f-3f01-4a66-968a-61add2b1f4ad.jpg



1340c01c-45a7-409a-ae40-3859c7aef6d5.jpg



bd3e3c87-7dc3-496a-ab6e-d1dc66424d3e.jpg

Películas

Hacemos cambios muy similares a los anteriores:

MoviesList.cs

```
@inject IJSRuntime js
@inject IRepository repository
@inject SweetAlertService swal

<div style="display:flex;flex-wrap:wrap;align-items:center;">
<GenericList List="Movies">
    <Loading>
        @Loading
    </Loading>
    <NoRecords>
        @NoRecords
    </NoRecords>
    <HasRecords Context="movie">
        <MovieItem Movie="movie" DeleteMovie="DeleteMovie"/>
    </HasRecords>
</GenericList>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

    private async Task DeleteMovie(Movie movie) {
        var confirmed = await js.Confirm($"Do you want to delete the movie '{movie.Title}'?");

        if (confirmed) {
            var responseHTTP = await repository.Delete($"api/movies/{movie.ID}");

            if (responseHTTP.Error) {
                var ErrMessage = await responseHTTP.GetErrorMessage();
                await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
            else {
                Movies!.Remove(movie);
            }
        }
    }
}
```

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
                .Take(limit)
                .OrderBy(movie => movie.ReleaseDate)
                .ToListAsync();

            var result = new HomePageDTO {
                OnBoard = onBoardMovies,
                NextReleases = nextReleases
            };
            return result;
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<MovieViewDTO>> Get(int id) {
            var movie = await context.Movies
```

```
.Where(movie => movie.ID == id)
.Include(movie => movie.GenresMovie)
.ThenInclude(gm => gm.Genre)
.Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
.ThenInclude(ma => ma.Actor)
.FirstOrDefaultAsync();

if (movie is null) {
    //No se encontró la película
    return NotFound();
}

//TODO: Sistema de votación.
var votesMedia = 4;
var userVote = 5;

var model = new MovieViewDTO();
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
```

```
model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, ".jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPut]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}
```

```
}
```



```
[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

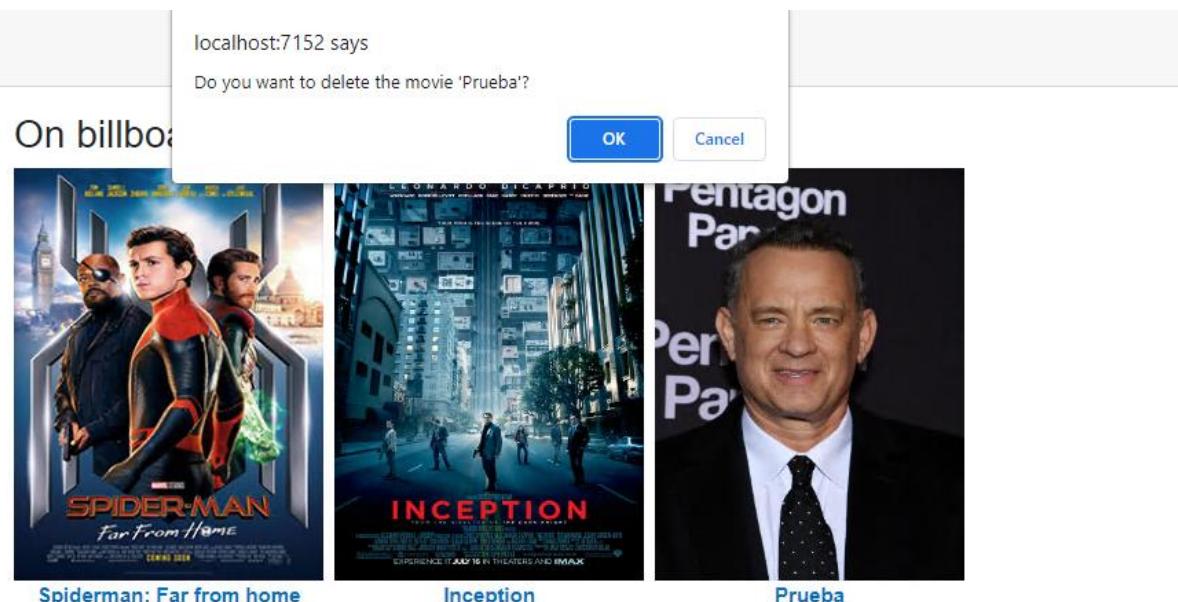
    if (movie is null)
        return NotFound();

    context.Remove(movie);    //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if (!string.IsNullOrWhiteSpace(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

    return NoContent();      //Todo se hizo correctamente
}
```

```
}
```

Probamos borrando la película Prueba. Antes de confirmar vemos que en la carpeta de **wwwroot/movies** del **Server** está la foto de Tom Hanks.



Next releases

No movies to show

rce > repos > BlazorPeliculas > Server > wwwroot > movies

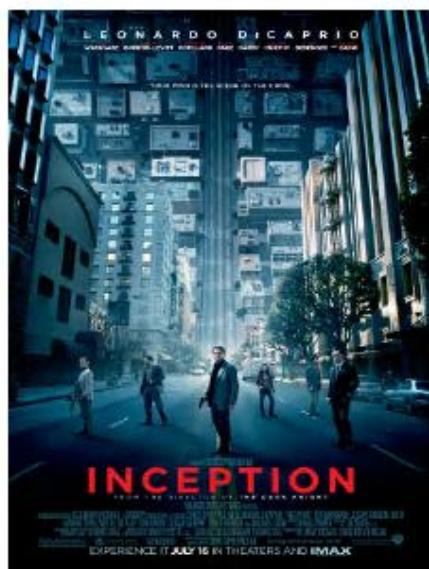


Al confirmar, desaparece la película del índice como así también el poster correspondiente:

On billboard



[Spiderman: Far from home](#)



[Inception](#)

[Edit](#)

[Delete](#)

[Edit](#)

[Delete](#)

Next releases

No movies to show

ce > repos > BlazorPeliculas > Server > wwwroot > movies



Paginación

Para conseguir la paginación creamos el archivo **PaginationDTO.cs** en la carpeta **DTOs** del proyecto **Shared**.

PaginationDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class PaginationDTO {
        public int Page { get; set; } = 1;
        public int RecordCount { get; set; } = 10;
    }
}
```

En **ActorsControllers.cs** agregamos el parámetro `pagination` del tipo **PaginationDTO** con el atributo **FromQuery** para obtener el dato complejo desde el **QueryString**.

Como la paginación la usaremos en distintas partes de la app centralizaremos el paginado. Creamos la clase **HttpContextExtensions.cs** en la carpeta **Helpers** del proyecto **Server**. La hacemos pública y estática. Creamos una tarea estática **InsertPaginationParametersInResponse** para poder informar al cliente, por ejemplo, cuántas páginas son y que éste pueda mostrar esa info en la interfaz del usuario.

Usaremos el tipo de dato **IQueryable** que es el que **Entity Framework** usa para permitirnos armar nuestro query. El método **FindFirstOrDefault** que venimos usando es un **IQueryable**.

El método `CountAsync` nos permite contar la cantidad de registros de una tabla (la que corresponda a `T`).

Con la línea:

```
await HttpContext.InserPaginationParametersInResponse(queryable, pagination.RecordCount);
```

de **ActorsController.cs** tenemos la metadata acerca de la cantidad de registros de la tabla. Para realizar la paginación, centralizaremos la operación creando la clase estática llamada **QueryableExtensions.cs** en la carpeta **Helpers** del proyecto **Server**. Con el método **Skip**, nos salteamos registros. Con el **Take** tomamos la cantidad de registros que queremos.

QueryableExtensions.cs

```
using BlazorPeliculas.Shared.DTOs;

namespace BlazorPeliculas.Server.Helpers {
    public static class QueryableExtensions {
        public static IQueryables<T> ToPage<T>(this IQueryables<T> queryable, PaginationDTO pagination)
        {
            return queryable
                .Skip((pagination.Page - 1) * pagination.RecordCount)
                .Take(pagination.RecordCount);
        }
    }
}
```

HttpContextExtensions.cs

```
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Helpers {
    public static class HttpContextExtensions {
        public async static Task InserPaginationParametersInResponse<T>(this HttpContext context,
            IQueryables<T> queryable, int amountRecordsToShow) {
            if(context is null)
                throw new ArgumentNullException(nameof(context));

            double count = await queryable.CountAsync();
            double totalPages = Math.Ceiling(count / amountRecordsToShow);

            context.Response.Headers.Add("count", count.ToString());
            context.Response.Headers.Add("totalPages", totalPages.ToString());
        }
    }
}
```

ActorsController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "people";

        public ActorsController(ApplicationDbContext context, IFileSaver fileSaver, IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Actor>>> Get([FromQuery] PaginationDTO pagination) {
            //return await context.Actors.ToListAsync();
            var queryable = context.Actors.AsQueryable();
            await HttpContext.InserPaginationParametersInResponse(queryable, pagination.RecordCount);
            return await queryable.OrderBy(x => x.Name).ToPage(pagination).ToListAsync();
        }

        [HttpGet("search/{searchText}")]
        public async Task<ActionResult<IEnumerable<Actor>>> Get(string searchText) {
            if(string.IsNullOrWhiteSpace(searchText))
                return new List<Actor>();

            searchText = searchText.ToLower();
            return await context.Actors
                .Where(x => x.Name.ToLower().Contains(searchText))
                .Take(5)
                .ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Actor>> Get(int id) {
            var actor = await context.Actors.FirstOrDefaultAsync(actor => actor.ID == id);

            if(actor is null)
```

```
return NotFound();

return actor;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Actor actor) {
    if(!string.IsNullOrWhiteSpace(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actor.Photo = await fileSaver.SaveFile(photoActor, "jpg", container);
    }

    context.Add(actor);
    await context.SaveChangesAsync();
    return actor.ID;
}

[HttpPut]
public async Task<ActionResult> Put(Actor actor) {
    var actorDB = await context.Actors.FirstOrDefaultAsync(a => a.ID == actor.ID);

    if(actorDB is null)
        return NotFound();
    //Tomá las propiedades de actor y pasalas a actorDB
    actorDB = mapper.Map(actor, actorDB);

    if(!string.IsNullOrWhiteSpace(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actorDB.Photo = await fileSaver.EditFile(photoActor, ".jpg", container, actorDB.Photo!);
    }

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var actor = await context.Actors.FirstOrDefaultAsync(x => x.ID == id);

    if(actor is null)
        return NotFound();

    context.Remove(actor);      //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if(!string.IsNullOrWhiteSpace(actor.Photo))
        await fileSaver.DeleteFile(actor.Photo!, container);
}
```

```

        return NoContent();
    }
}
}

```

Probemos la paginación. Vamos al listado de actores y vemos que son 4.

localhost:7152/actors

Name
Joseph Gordon-Levitt
Leonardo DiCaprio
Tom Hanks
Tom Holland

De hecho, si vamos a <https://localhost:7152/api/actors> obtenemos (acorté las bios para que no sea tan largo):

```

[
  {
    "id":4,
    "name":"Joseph Gordon-Levitt",
    "bio":"Joseph Leonard Gordon-Levitt was born February 17, 1981 in Los Angeles...",
    "photo":"https://localhost:7152/people/26add85f-3f01-4a66-968a-61add2b1f4ad.jpg",
    "birthDate":"1981-02-17T00:00:00",
    "character":null,
    "movieActor":[
      ]
  },
  {
    "id":3,
    "name":"Leonardo DiCaprio",
    "bio":"Leonardo Wilhelm DiCaprio is an American actor, producer, and environmental activist. He is one of the most recognizable faces in Hollywood and has won numerous awards and nominations for his work in film, including an Academy Award for Best Supporting Actor for his role in 'The Departed'. DiCaprio's career spans over two decades, with hits like 'The Wolf of Wall Street', 'Inception', and 'The Revenant'. He is also known for his activism, particularly regarding climate change, and has been involved in various environmental initiatives.", "photo":"https://localhost:7152/people/26add85f-3f01-4a66-968a-61add2b1f4ad.jpg",
    "birthDate":"1974-08-11T00:00:00",
    "character":null,
    "movieActor": [
      ]
  }
]

```

```

"bio":"Few actors in the world have had a career quite as diverse as Leonardo DiCaprio's...",
"photo":"https://localhost:7152/people/0c994f63-e7e8-4e45-9619-06121b1bb141.jpg",
"birthDate":"1974-11-11T00:00:00",
"character":null,
"movieActor":[
    ]
},
{
    "id":2,
    "name":"Tom Hanks",
    "bio":"### Thomas Jeffrey Hanks\n\nHe was born July 9, 1956. He is an American actor and filmmaker...",
    "photo":"https://localhost:7152/people/1340c01c-45a7-409a-ae40-3859c7aef6d5.jpg",
    "birthDate":"1956-07-09T00:00:00",
    "character":null,
    "movieActor":[
        ]
},
{
    "id":1,
    "name":"Tom Holland",
    "bio":"### Thomas Stanley Holland\n\nHe was born 1 June 1996. He is an English actor...",
    "photo":"https://localhost:7152/people/2b89d596-5f00-4ab3-9890-f2e853fbbe96.jpg",
    "birthDate":"1996-06-01T00:00:00",
    "character":null,
    "movieActor":[
        ]
}
]
    
```

En el listado anterior vinieron los IDs 4, 3, 2 y 1.

Pero si vamos a <https://localhost:7152/api/actors?RecordCount=2> obtenemos:

```

[
{
    "id":4,
    "name":"Joseph Gordon-Levitt",
    "bio":"Joseph Leonard Gordon-Levitt was born February 17, 1981 in Los Angeles...",
    "photo":"https://localhost:7152/people/26add85f-3f01-4a66-968a-61add2b1f4ad.jpg",
    "birthDate":"1981-02-17T00:00:00",
    "character":null,
    "movieActor":[
        ]
},
{
    "id":3,
    "name":"Leonardo DiCaprio",
    "bio":"Few actors in the world have had a career quite as diverse as Leonardo DiCaprio's...",
    "photo":"https://localhost:7152/people/0c994f63-e7e8-4e45-9619-06121b1bb141.jpg",
    "birthDate":"1974-11-11T00:00:00",
    "character":null,
    "movieActor":[
        ]
}
    
```

Obteniendo sólo los primeros 2 registros (IDs 4 y 3) tal como se solicitó. Sin embargo, si vamos a a <https://localhost:7152/api/actors?RecordCount=2&Page=2> obtenemos:

```
[
  {
    "id":2,
    "name":"Tom Hanks",
    "bio": "### Thomas Jeffrey Hanks\n\nHe was born July 9, 1956. He is an American actor and filmmaker...",
    "photo":"https://localhost:7152/people/1340c01c-45a7-409a-ae40-3859c7aef6d5.jpg",
    "birthDate":"1956-07-09T00:00:00",
    "character":null,
    "movieActor":[
      ]
  },
  {
    "id":1,
    "name":"Tom Holland",
    "bio": "### Thomas Stanley Holland\n\nHe was born 1 June 1996. He is an English actor...",
    "photo":"https://localhost:7152/people/2b89d596-5f00-4ab3-9890-f2e853fbbe96.jpg",
    "birthDate":"1996-06-01T00:00:00",
    "character":null,
    "movieActor":[
      ]
  }
]
```

Es decir que salteo los primeros 2 y mostró los segundos 2 (IDs 2 y 1).

Para crear un componente de paginación utilizaremos Bootstrap que ya tiene resuelto esto en <https://getbootstrap.com/docs/5.3/components/pagination/>. Agregamos el componente **Pagination.razor** en la carpeta **Shared** de nuestro proyecto **Client**.

Radio significa cuántas páginas posteriores y anteriores se listan. Por ejemplo, un radio de 3 mientras que visualizamos la página 8 sería:

Previous	5	6	7	8	9	10	11	Next
--------------------------	-------------------	-------------------	-------------------	-------------------	-------------------	--------------------	--------------------	----------------------

Pagination.razor

```
@page "/actors"
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl
<h3>Actors list</h3>

<div class="mb-3">
  <a href="actors/create" class="btn btn-info">Create actor</a>
</div>
```

```

<Pagination ActualPage="ActualPage"
    TotalPages="TotalPages"
    SelectedPage="SelectedPage" />

<GenericList List="Actors">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Actors!) {
                    <tr>
                        <td>
                            <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                            <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                        </td>
                        <td>&ampnbsp@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }
    private int ActualPage = 1;
    private int TotalPages;

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task SelectedPage(int page) {
        ActualPage = page;
        await Load(page);
    }

    private async Task Load(int Page = 1) {
        var responseHTTP = await repository.Get<List<Actor>>($"api/actors?page={Page}");
        Actors = responseHTTP.Response!;
        TotalPages =
int.Parse(responseHTTP.httpResponseMessage.Headers.GetValues("totalPages").FirstOrDefault());
    }

    public async Task Delete(Actor actor) {
        var responseHTTP = await repository.Delete($"api/actors/{actor.ID}");
    }
}

```

```

if (responseHTTP.Error) {
    if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
        navManager.NavigateTo("/");
    else {
        var ErrMessage = await responseHTTP.GetErrMsg();
        await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
    }
}
else {
    await Load();
}
}
}

```

La cantidad de páginas totales las recuperamos del header que agrega la clase **HttpContextExtensions**.

ActorsList.razor

```

@page "/actors"
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl
<h3>Actors list</h3>
<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>
<Pagination ActualPage="ActualPage"
    TotalPages="TotalPages"
    SelectedPage="SelectedPage" />
<GenericList List="Actors">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Actors!) {
                    <tr>
                        <td>
                            <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                            <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                        </td>
                        <td>&nbsp;@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>

```

```
</tr>
}
</tbody>
</table>
</HasRecordsComplete>
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }
    private int ActualPage = 1;
    private int TotalPages;

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task SelectedPage(int page) {
        ActualPage = page;
        await Load(page);
    }

    private async Task Load(int Page = 1) {
        var responseHTTP = await repository.Get<List<Actor>>($"api/actors?page={Page}");
        Actors = responseHTTP.Response!;
        TotalPages =
            int.Parse(responseHTTP.httpResponseMessage.Headers.GetValues("totalPages").FirstOrDefault()!);
    }

    public async Task Delete(Actor actor) {
        var responseHTTP = await repository.Delete($"api/actors/{actor.ID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            await Load();
        }
    }
}
```

Momentáneamente ponemos el **RecordCount** de **PaginationDTO** en 2 para ver la paginación:

Actors list

[Create actor](#)[Previous](#) [1](#) [2](#) [Next](#)**Name**[Edit](#)[Delete](#)

Joseph Gordon-Levitt

[Edit](#)[Delete](#)

Leonardo DiCaprio

Actors list

[Create actor](#)[Previous](#) [1](#) [2](#) [Next](#)**Name**[Edit](#)[Delete](#)

Tom Hanks

[Edit](#)[Delete](#)

Tom Holland

Filtros

Para poder filtrar, creamos la clase **SearchMoviesParametersDTO** en la carpeta **DTOs** del proyecto **Shared**. Creamos los campos **Page** y **RecordCount** para recibirlas desde el **QueryString** y luego cargarlos directamente en el **PaginationDTO**.

SearchMoviesParametersDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class SearchMoviesParametersDTO {
        public int Page { get; set; } = 1;
        public int RecordCount { get; set; } = 10;
        public PaginationDTO pagination {
            get {
                return new PaginationDTO {
                    Page = Page,
                    RecordCount = RecordCount
                };
            }
        }

        public string? Title { get; set; }
        public int GenreID { get; set; }
        public bool Onbillboard { get; set; }
        public bool Releases { get; set; }
        public bool MostVoted { get; set; }
    }
}
```

Ejecución diferida

En **MoviesController** utilizaremos lo que se conoce en **Entity Framework Core** como ejecución diferida. Básicamente, iremos armando la query paso por paso. Iremos aplicando filtros de manera opcional. Recordamos que un **Queryable** en **Entity Framework Core** es un tipo de dato que representa ese query que vamos a ir armando, parte por parte.

Para el filtro de géneros hago una primera proyección con **GenresMovie** que es el listado de los géneros que están seleccionados. Con él, nos quedamos sólo aquellos en los que alguno de ellos sea el género seleccionado.

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
```

```
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
                .Take(limit)
                .OrderBy(movie => movie.ReleaseDate)
                .ToListAsync();

            var result = new HomePageDTO {
                OnBoard = onBoardMovies,
                NextReleases = nextReleases
            };
            return result;
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<MovieViewDTO>> Get(int id) {
            var movie = await context.Movies
                .Where(movie => movie.ID == id)
                .Include(movie => movie.GenresMovie)
                .ThenInclude(gm => gm.Genre)
        }
    }
}
```

```

.Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
.ThenInclude(ma => ma.Actor)
.FirstOrDefaultAsync();

if (movie is null) {
    //No se encontró la película
    return NotFound();
}

//TODO: Sistema de votación.
var votesMedia = 4;
var userVote = 5;

var model = new MovieViewDTO();
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("filter")]
public async Task<ActionResult<List<Movie>>> Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);

    if(model.Releases) {
        var today = DateTime.Today;

        queryableMovies = queryableMovies
            .Where(x => x.ReleaseDate >= today);
    }

    if (model.GenreID != 0)

```

```
queryableMovies = queryableMovies
    .Where(x => x.GenresMovie
        .Select(y => y.GenreID)
        .Contains(model.GenreID));

    //TODO: Implementar votación

    await HttpContext.InsertPaginationParametersInResponse(queryableMovies,
model.RecordCount);

    //Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
    var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
    return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, ".jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}
```

```
}
```

```
private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPost]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}
```

```
[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if (movie is null)
        return NotFound();

    context.Remove(movie); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if (!string.IsNullOrWhiteSpace(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

    return NoContent();   //Todo se hizo correctamente
}
```

{
}

En el componente **FilterMovie** tenemos el nuevo método **GenerateQueryStrings**. En el concatenamos todos los parámetros que no estén en el array de valores por defecto. Primero preguntamos por los elementos del diccionario cuyo x no esté en el listado de valores por defecto. De ese pre-filtro, hacemos una proyección donde para cada x armamos un string del tipo **Key=Value** donde Value se encodea para que sea compatible con una URL. Para finalizar lo devuelve como array porque la función Join concatena todos los ítems de un array en un string.

Para que el combo de género se setee automáticamente de acuerdo con el **QueryString**, cambiamos el campo Genre a que sea un parámetro con el atributo **SupplyParameterFromQuery**. Lo mismo para todos los demás.

También agregamos el componente de paginación.

FilterMovie.razor

```
@page "/movies/filter"
@inject IRepository repository
@inject NavigationManager navManager
<h3>Movies filter</h3>

<div class="row g-3 align-items-center mb-3">
    <div class="col-sm-3">
        <input type="text" class="form-control" id="title" placeholder="Movie title"
            @bind-value="Title" @bind-value:event="oninput"
            @onkeypress="@((KeyboardEventArgs e) => TitleKeyPress(e))" />
    </div>
    <div class="col-sm-3">
        <select class="form-select" @bind="Genre">
            <option value="0">-- Select a genre --</option>
            @foreach(var item in genres) {
                <option value="@item.ID">@item.Name</option>
            }
        </select>
    </div>
    <div class="col-sm-6" style="display:flex;">
        <div class="form-check me-2">
            <input type="checkbox" class="form-cheq-input" id="premieres" @bind="futurePremieres" />
            <label class="form-check-label" for="premieres">Future premieres</label>
        </div>
        <div class="form-check me-2">
            <input type="checkbox" class="form-cheq-input" id="billboard" @bind="onBillboard" />
            <label class="form-check-label" for="billboard">On billboard</label>
        </div>
        <div class="form-check">
            <input type="checkbox" class="form-cheq-input" id="mostVoted" @bind="mostVoted" />
            <label class="form-check-label" for="mostVoted">Most voted</label>
        </div>
    </div>
</div>
```

```

        </div>
    </div>

<div class="col-12">
    <button type="button" class="btn btn-primary" @onclick="FilteredMovies">Filter</button>
    <button type="button" class="btn btn-danger" @onclick="Clean">Clean</button>
</div>
</div>

<Pagination ActualPage="actualPage" TotalPages="totalPages" SelectedPage="SelectedPage" />

<MoviesList Movies="Movies" ></MoviesList>
@code {
    [Parameter, SupplyParameterFromQuery] public string Title { get; set; } = "";
    [Parameter, SupplyParameterFromQuery(Name = "genreid")] public int Genre { get; set; } = 0;
    private List<Genre> genres = new List<Genre>();
    [Parameter, SupplyParameterFromQuery] public bool futurePremieres { get; set; } = false;
    [Parameter, SupplyParameterFromQuery] public bool onBillboard { get; set; } = false;
    [Parameter, SupplyParameterFromQuery] public bool mostVoted { get; set; } = false;
    private List<Movie>? Movies;
    Dictionary<string, string> queryStringDict = new Dictionary<string, string>();
    [Parameter, SupplyParameterFromQuery] public int actualPage { get; set; } = 1;
    private int totalPages { get; set; } = 1;

    protected override async Task OnInitializedAsync() {
        //Movies = repository.GetMovies();
        if(actualPage == 0) actualPage = 1; //Por las dudas de que en el queryString viniera un 0.
        await GetGenres();
        await PerformFilter(GenerateQueryStrings());
    }

    private async Task SelectedPage(int page) {
        actualPage = page;
        await FilteredMovies();
    }

    private async Task GetGenres() {
        var responseHTTP = await repository.Get<List<Genre>>("api/genres");
        genres = responseHTTP.Response;
    }

    private async Task TitleKeyPress(KeyboardEventArgs e)
    {
        if(e.Key == "Enter") {
            await FilteredMovies();
        }
    }

    private async Task FilteredMovies() {
        var queryString = GenerateQueryStrings();

```

```

        navManager.NavigateTo($""/movies/filter?{queryString}");
        await PerformFilter(queryString);
    }

    private async Task PerformFilter(string queryString) {
        var responseHTTP = await repository.Get<List<Movie>>($"api/movies/filter?{queryString}");
        Movies = responseHTTP.Response;
        totalPages =
            int.Parse(responseHTTP.httpResponseMessage.Headers.GetValues("totalPages").FirstOrDefault()!);
    }

    private string GenerateQueryStrings() {
        if(queryStringDict is null)
            queryStringDict = new Dictionary<string, string>();

        queryStringDict["genreid"] = Genre.ToString();
        queryStringDict["title"] = Title ?? string.Empty;
        queryStringDict["futurePremieres"] = futurePremieres.ToString();
        queryStringDict["onBillboard"] = onBillboard.ToString();
        queryStringDict["mostVoted"] = mostVoted.ToString();
        queryStringDict["page"] = actualPage.ToString();

        var defaultValues = new List<string>() { "false", "", "0" };

        return string.Join("&", queryStringDict.Where(x =>
            !defaultValues.Contains(x.Value.ToLower()))
            .Select(x => $"{x.Key}={System.Web.HttpUtility.UrlEncode(x.Value)}")
            .ToArray());
    }

    private async Task Clean() {
        Title = "";
        Genre = 0;
        futurePremieres = false;
        onBillboard = false;
        mostVoted = false;
        await FilteredMovies();
    }
}

```

Algoritmo de Diferencias y @Key

Blazor utiliza un algoritmo de diferencias para actualizar eficientemente el DOM (Document Object Model) de nuestra app.

Cada vez que actualizamos, por ejemplo, una variable, eso se puede reflejar en el UI (User Interface). Sin embargo, para eso, para no estar reescribiendo toda la página, Blazor utiliza el algoritmo de diferencias que le permite actualizar la página

de una manera súper, super eficiente. Este algoritmo es genérico y por tanto, a veces necesita que le ayudemos con algunas tareas específicas.

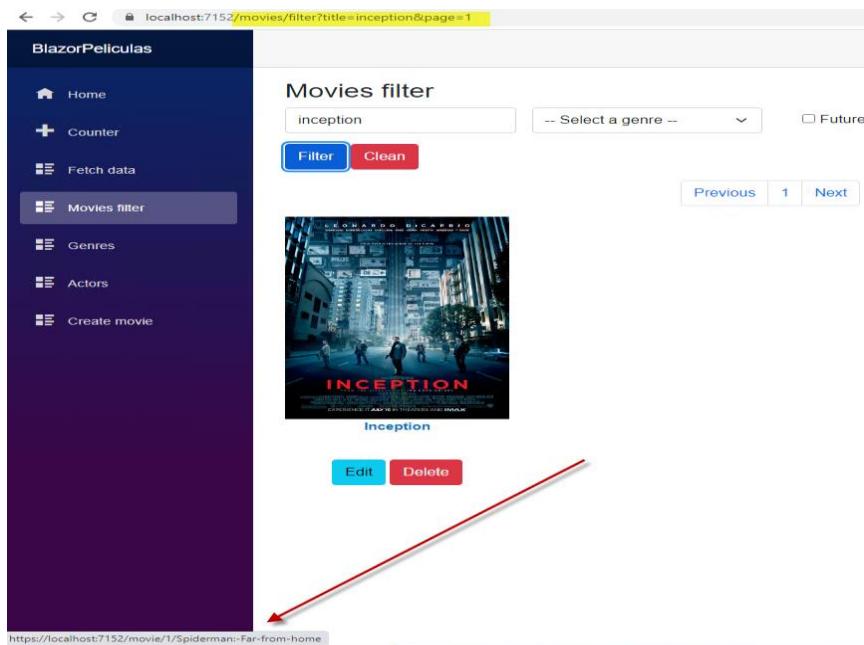
Por ejemplo, en el listado de películas tenemos dos películas Spiderman e Inception.

The screenshot shows the BlazorPelículas application's home page. On the left is a sidebar with a dark purple background containing navigation links: Home, Counter, Fetch data, Movies filter, Genres, Actors, and Create movie. The main content area has a light gray background. At the top, it says "On billboard". Below this are two movie cards. The first card is for "Spiderman: Far from home" and the second for "Inception". Each card has a small thumbnail, the movie title, and "Edit" and "Delete" buttons at the bottom. Below the cards, there is a section titled "Next releases" with the message "No movies to show".

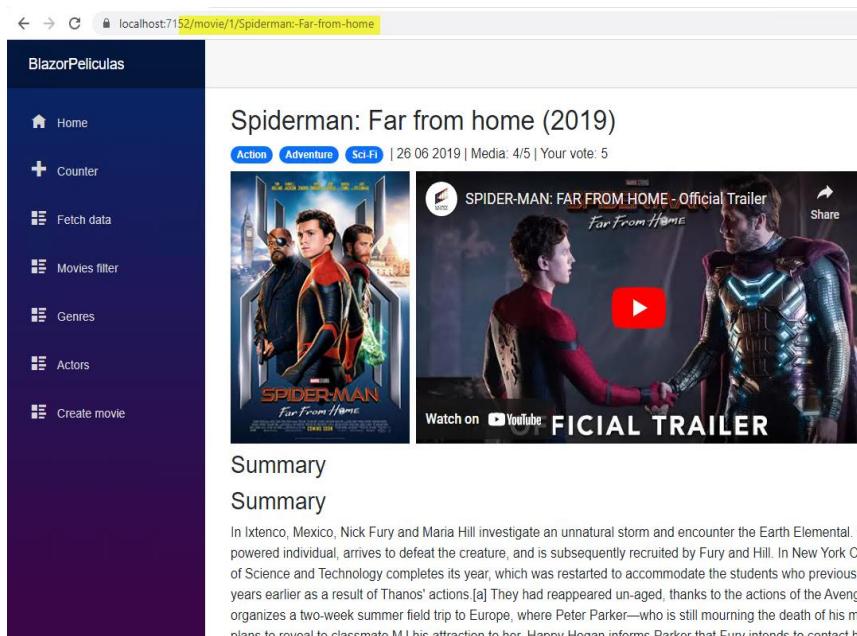
Si le damos click a Spiderman, vamos a ver que caemos en la página de visualización de Spiderman.

The screenshot shows the BlazorPelículas application's movie details page for "Spiderman: Far from home (2019)". The sidebar on the left is identical to the previous screenshot. The main content area shows the movie title "Spiderman: Far from home (2019)" with genre tags (Action, Adventure, Sci-Fi), release date (26 06 2019), media rating (4/5), and user vote (5). Below this are two images: the movie poster on the left and a trailer thumbnail on the right. The trailer thumbnail includes a play button and the text "Watch on YouTube OFFICIAL TRAILER". Below the images, there is a section titled "Summary" with a short description of the plot involving an unnatural storm and the Earth Elemental.

Ahora, si vamos a Movies filter y filtramos la película Inception y presionamos Filter. El filtro muestra sólo inception en el listado, pero si nos paramos sobre la foto podemos ver el link al que apunta:



De hecho, si clickeamos en la foto volvemos a caer en Spiderman;



Lo que pasa es que Blazor no tiene forma de saber la diferencia entre Spiderman e Inception. Es decir, a nivel de HTML esto es una colección de elementos de HTML, pero Blazor no sabe distinguir la diferencia entre uno y otro.

Y entonces cuando carga el listado de películas al principio Blazor “memoriza” el HTML de ese listado (Spiderman es la película 1 e Inception la 2). Cuando vamos a filtro y pedimos por Inception, pues Blazor ve que ya no hay dos elementos, sino que hay solamente uno, pero él no sabe que este primero que está aquí es diferente al primer elemento que estaba anteriormente que era Spiderman.

Para ayudarlo, podemos usar el atributo **Key** que me va a permitir indicarle a Blazor como diferenciar los elementos de una lista. Básicamente, en el listado de películas indicamos cuál es el **Key** de cada elemento. Con eso alcanza para que Blazor sepa como diferenciar entre un item y otro.

FilterMovie.razor

```
@inject IJSRuntime js
@inject IRepository repository
@inject SweetAlertService swal

<div style="display:flex;flex-wrap:wrap;align-items:center;">
<GenericList List="Movies">
<Loading>
    @Loading
</Loading>
<NoRecords>
    @NoRecords
</NoRecords>
<HasRecords Context="movie">
    <MovieItem Movie="movie"
        DeleteMovie="DeleteMovie"
        @key="movie.ID"/>
</HasRecords>
</GenericList>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

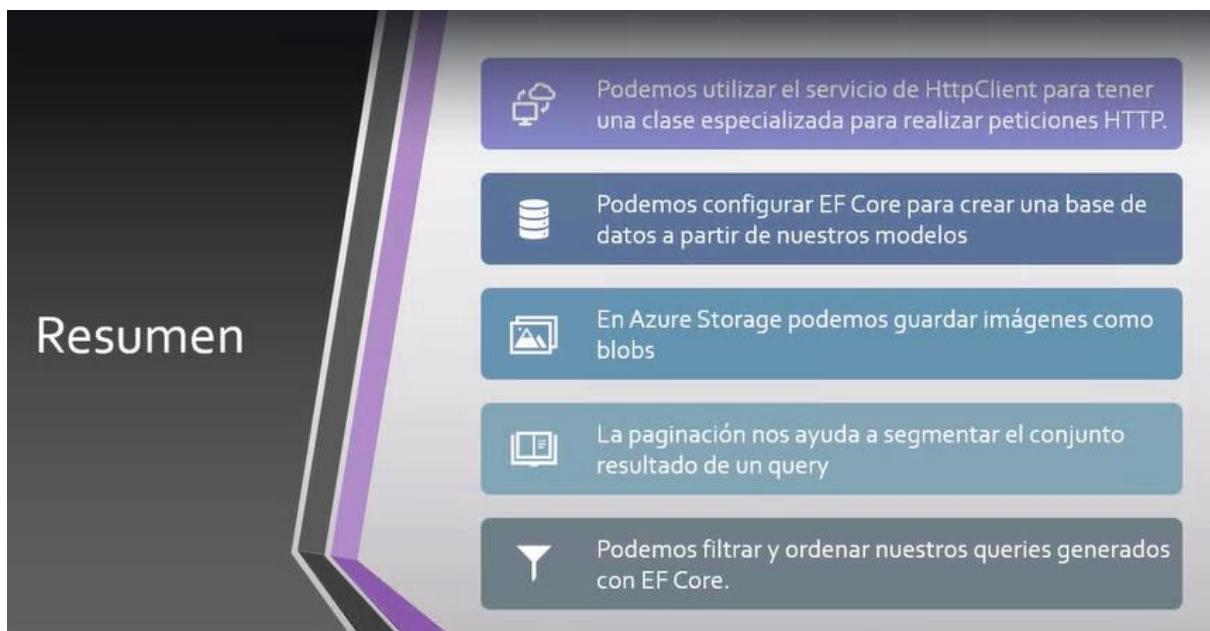
    private async Task DeleteMovie(Movie movie) {
        var confirmed = await js.Confirm($"Do you want to delete the movie '{movie.Title}'?");

        if (confirmed) {
            var responseHTTP = await repository.Delete($"api/movies/{movie.ID}");

            if (responseHTTP.Error) {
                var ErrMessage = await responseHTTP.GetErrorMessage();
            }
        }
    }
}
```

```
        await swal.fireAsync("Error", ErrMessage, SweetAlertIcon.Error);
    }
    else {
        Movies!.Remove(movie);
    }
}
}
```

Con ese cambio alcanza para resolver el bug mostrado recién.



Seguridad

Crearemos un sistema de usuarios para que sólo algunos puedan agregar y, sobre todo, borrar películas, por ejemplo.

Para ello, es necesario agregar el paquete de **NuGet** en el proyecto **Client**:

Microsoft.AspNetCore.Components.WebAssembly.Authentication

Es una buena práctica ir a la solapa Updates y bajar las actualizaciones de todos los paquetes que tengamos instalados.

Es necesario agregar el using en el archivo _Imports.razor

```
_Imports.razor
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
```

```
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorPeliculas.Client
@using BlazorPeliculas.Client.Shared
@using BlazorPeliculas.Client.Utilities
@using BlazorPeliculas.Shared.Entities
@using BlazorPeliculas.Client.Repositories
@using BlazorPeliculas.Client.Helpers
@using CurrieTechnologies.Razor.SweetAlert2
@using BlazorPeliculas.Shared.DTOs
@using Microsoft.AspNetCore.Components.Authorization
```

Autenticación

Creamos la carpeta **Auth** en el proyecto **Client** y en ella una clase llamada **AuthProviderTest** que será un proveedor de autenticación. La hacemos implementar **AuthenticationStateProvider**.

Un proveedor de autenticación le informa a Blazor sobre el estado de autenticación de una persona (¿sabemos quien es?).

Autenticación no es lo mismo que **autorización**. La primera responde a quién eres, mientras que la segunda responde a qué puedes hacer.

Un claim es simplemente un dato acerca del usuario como su nombre, fecha de nacimiento, email, etc. Como este es un usuario anónimo, básicamente no le pondremos ningún dato.

El método **GetAuthenticationStateAsync** es ejecutado inmediatamente se inicia nuestra app.

AuthProviderTest.cs

```
using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class AuthProviderTest : AuthenticationStateProvider {
        public override async Task<AuthenticationState> GetAuthenticationStateAsync() {
            var anonymous = new ClaimsIdentity();
            return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonymous)));
        }
    }
}
```

Abrimos la clase **Program** del proyecto **Client** para configurar las dependencias. Agregamos el servicio e indicamos cuál es nuestro proveedor de autenticación.

Otra cosa que tenemos que cambiar es que el servicio de **HttpClient** sea **Singleton** porque en el futuro vamos a enviar lo que se conoce como un JSON web token a través del **HttpClient**.

Por lo tanto, necesitamos que sea un singleton para que así, cuando configuremos una instancia del **HttpClient**, esa configuración se propague por toda la aplicación.

Así vamos a poder mandar ese JSON web token, que no es más que un pequeño string que identifica al usuario para que así pueda autenticarse en nuestro proyecto de **Server**. **Client** y **Server** se van a ejecutar en ambientes distintos: El proyecto **Client** se ejecuta en el navegador del usuario, mientras que el proyecto **Server** se ejecuta en un servidor web.

Entonces a través de ese JSON web token lo vamos a utilizar a través de nuestro http client.

Program.cs

```
using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Repositories;
using CurrieTechnologies.Razor.SweetAlert2;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });
configureServices(builder.Services);
await builder.Build().RunAsync();

void configureServices(IServiceCollection services) {
    services.AddScoped< IRepository, Repository>(); /* Inyectar un IRepository.
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
    clase Repository. */
    services.AddSweetAlert2();
    services.AddAuthorizationCore();
    services.AddScoped<AuthenticationStateProvider, AuthProviderTest>();
}
```

Para poder disponer de los datos del usuario autenticado utilizado cambiamos el componente **RouteView** por **AuthorizeRouteView** en el componente **App.razor**. Para

el componente **NotFound**, en cambio, utilizamos un parámetro de cascada. El **Framework** de Blazor tiene el **CascadingAuthorizationState**.

App.razor

```
@using Microsoft.AspNetCore.Components.WebAssembly.Services;
@using System.Reflection;
@inject LazyAssemblyLoader lazyLoader

<Router AppAssembly="@typeof(App).Assembly"
    OnNavigateAsync="OnNavigateAsync"
    AdditionalAssemblies="assemblies">
    <Found Context="routeData">
        <AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)" />
        <FocusOnNavigate RouteData="@routeData" Selector="h1" />
    </Found>
    <NotFound>
        <CascadingAuthenticationState>
            <PageTitle>Not found</PageTitle>
            <LayoutView Layout="@typeof(MainLayout)">
                <p role="alert">Sorry, there's nothing at this address.</p>
            </LayoutView>
        </CascadingAuthenticationState>
    </NotFound>
</Router>

@code {
    private List<Assembly> assemblies = new List<Assembly>();
    private async Task OnNavigateAsync(NavigationContext args) {
        if(args.Path.EndsWith("counter")) {
            var loadedAssemblies = await lazyLoader.LoadAssembliesAsync(
                new List<string> { "MathNet.Numerics.dll" }
            );
            assemblies.AddRange(loadedAssemblies);
        }
    }
}
```

Si ejecutáramos no veríamos ningún cambio porque el usuario, por el momento, es anónimo. Simularemos una demora de 3 segundos (como si estuviera intentando autenticar a un usuario contra un proveedor).

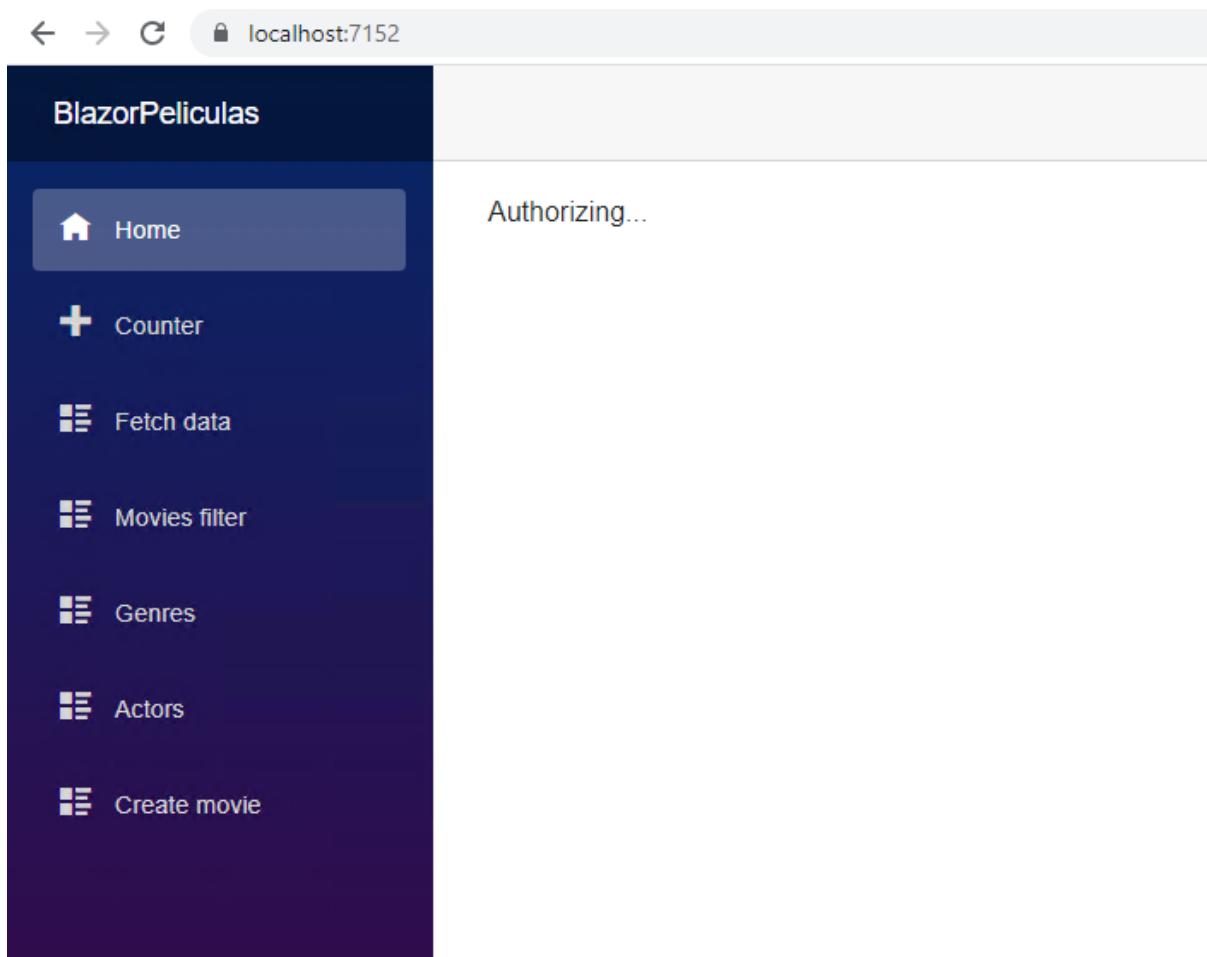
AuthProviderTest.cs

```
using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class AuthProviderTest : AuthenticationStateProvider {
        public override async Task<AuthenticationState> GetAuthenticationStateAsync() {
            await Task.Delay(3000);
            var anonymous = new ClaimsIdentity();
```

```
        return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonymous)));
    }
}
```

Ahora, vemos que Blazor está indicando, a través del componente `AuthorizeRouteView` que se está ejecutando la autenticación:



Si queremos personalizar el mensaje mientras que se está autorizando al usuario, usamos el **RenderFragment Authorizing** del **AuthorizationRouteView**:

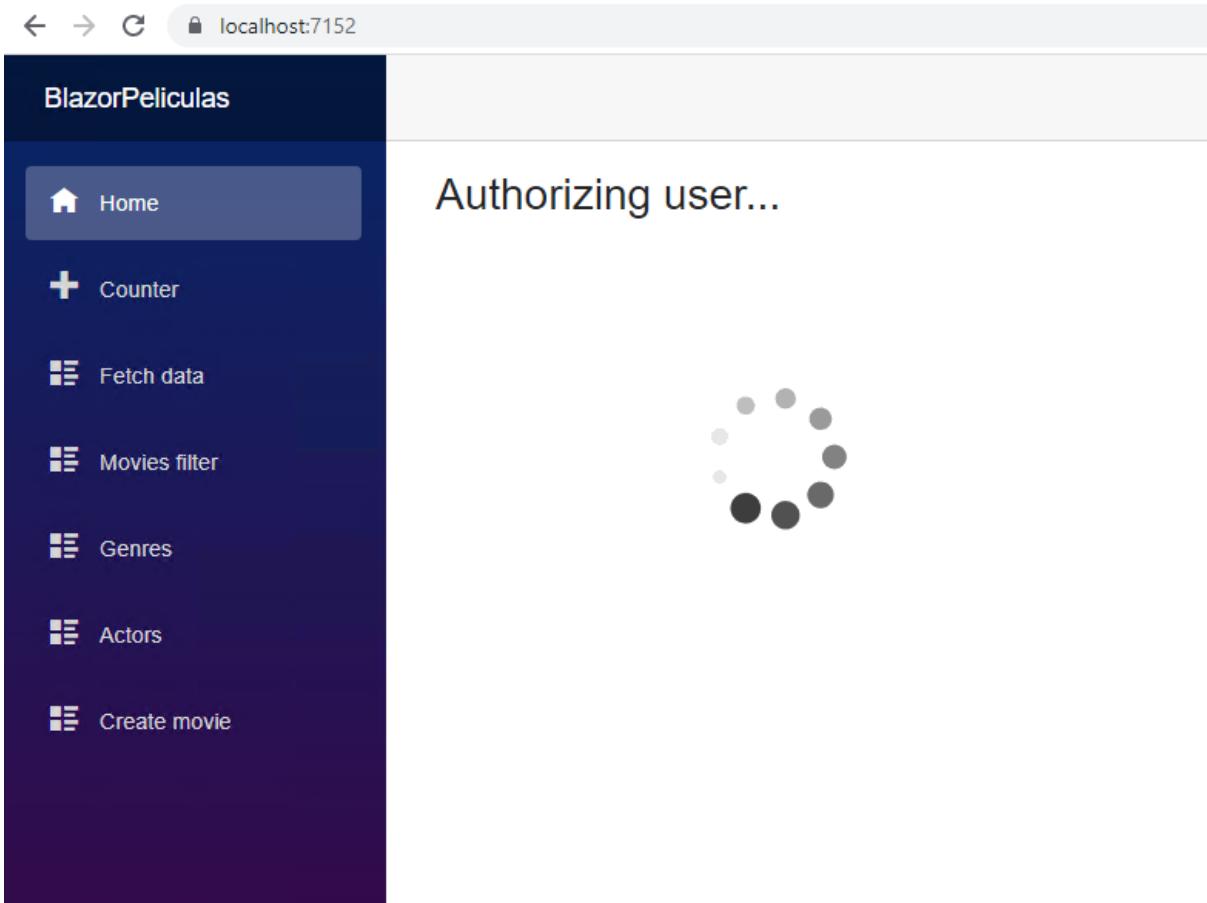
App.razor

```
@using Microsoft.AspNetCore.Components.WebAssembly.Services;
@using System.Reflection;
@inject LazyAssemblyLoader lazyLoader

<Router AppAssembly="@typeof(App).Assembly"
OnNavigateAsync="OnNavigateAsync"
AdditionalAssemblies="assemblies">
<Found Context="routeData">
```

```
<AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">
    <Authorizing>
        <h3>Authorizing user...</h3>
        <LoadingWheel/>
    </Authorizing>
</AuthorizeRouteView>
<FocusOnNavigate RouteData="@routeData" Selector="h1" />
</Found>
<NotFound>
    <CascadingAuthenticationState>
        <PageTitle>Not found</PageTitle>
        <LayoutView Layout="@typeof(MainLayout)">
            <p role="alert">Sorry, there's nothing at this address.</p>
        </LayoutView>
    </CascadingAuthenticationState>
</NotFound>
</Router>

@code {
    private List<Assembly> assemblies = new List<Assembly>();
    private async Task OnNavigateAsync(NavigationContext args) {
        if(args.Path.EndsWith("counter")) {
            var loadedAssemblies = await lazyLoader.LoadAssembliesAsync(
                new List<string> { "MathNet.Numerics.dll" }
            );
            assemblies.AddRange(loadedAssemblies);
        }
    }
}
```



AuthorizeView

Utilizaremos el componente **AuthorizeView** para mostrar/ocultar elementos según el estado de autenticación. Este nos permite mostrar sólo si estamos autenticados. Por ejemplo, el siguiente código de Index.razor no muestra nada nuevo porque no estamos autenticados (anónimo):

Index.razor

```
@page "/"
@inject IRepository repository

<PageTitle>Blazor Movies</PageTitle>
<AuthorizeView>
    <p>You are authenticated...</p>
</AuthorizeView>
<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
```

```

</Loading>
<NoRecords>
    <p>No movies to show</p>
</NoRecords>
<MoviesList>
</div>
</div>
<h3>Next releases</h3>
<div>
    <MoviesList Movies="NextReleases">
        <Loading>
            
        </Loading>
        <NoRecords>
            <p>No movies to show</p>
        </NoRecords>
    </MoviesList>
</div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
        OnBoard = responseHTTP.Response!.OnBoard;
        NextReleases = responseHTTP.Response!.NextReleases;
    }
}

```

Sin embargo, si nuestra clase de proveedor de autenticación devolviera que hay un usuario logueado:

AuthProviderTest.cs

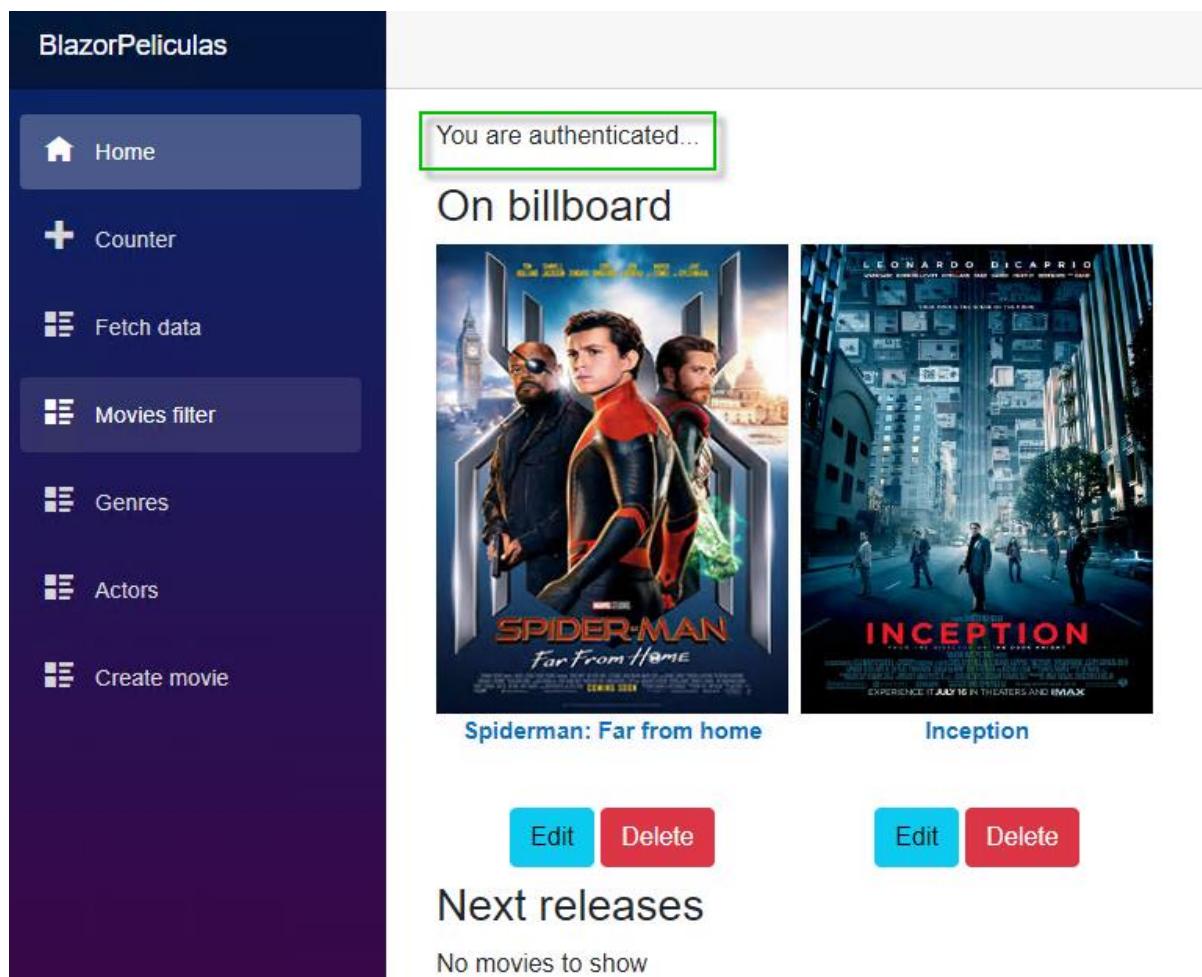
```

using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class AuthProviderTest : AuthenticationStateProvider {
        public override async Task<AuthenticationState> GetAuthenticationStateAsync() {
            var anonymous = new ClaimsIdentity();
            var user = new ClaimsIdentity(authenticationType: "test");
            return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(user)));
        }
    }
}

```

Veremos que el componente AuthorizeView ya nos muestra el párrafo de "You're authenticated".



Podemos mostrar contenido si no estamos autorizados también. Si volvemos al usuario anónimo:

AuthProviderTest.cs

```
using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class AuthProviderTest : AuthenticationStateProvider {
        public override async Task<AuthenticationState> GetAuthenticationStateAsync() {
            var anonymous = new ClaimsIdentity();
            var user = new ClaimsIdentity(authenticationType: "test");
            return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(anonymous)));
        }
    }
}
```

Y agregamos 2 **RenderFragment** en **Index.razor**:

Index.razor

```
@page "/"
@inject IRepository repository

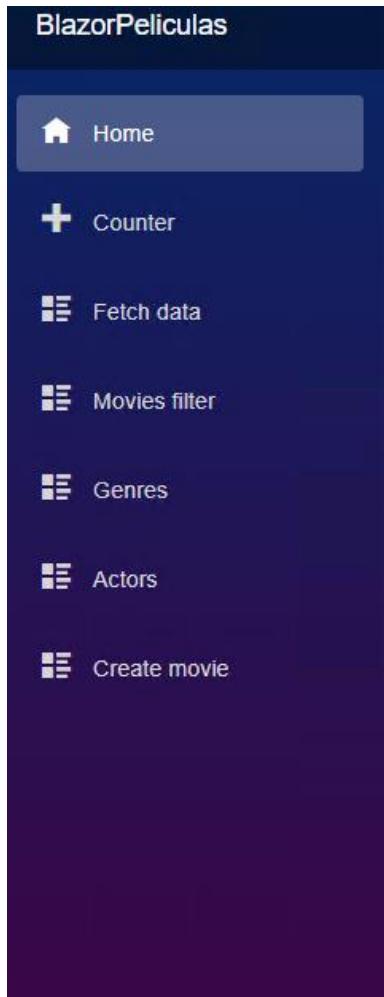
<PageTitle>Blazor Movies</PageTitle>
<AuthorizeView>
    <Authorized>
        <p>You are authenticated...</p>
    </Authorized>
    <NotAuthorized>
        <p>You are NOT authenticated...</p>
    </NotAuthorized>
</AuthorizeView> <div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync()
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");}
```

```

        OnBoard = responseHTTP.Response!.OnBoard;
        NextReleases = responseHTTP.Response!.NextReleases;
    }
}
    
```

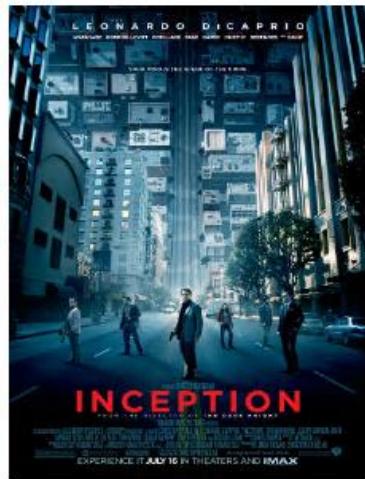


You are NOT authenticated...

On billboard



[Spiderman: Far from home](#)



[Inception](#)

[Edit](#) [Delete](#)

[Edit](#) [Delete](#)

Next releases

No movies to show

Claims es como la cédula o el DNI de una persona. Es emitida por un organismo gubernamental en el cuál podemos confiar. Un claim puede venir desde muchos lugares: Facebook, Twitter, tu BD, etc.

En nuestra clase proveedor, podemos establecer una lista de Claims que son pares de Key+Value. En el siguiente ejemplo, usamos el **ClaimTypes.Name** para poder utilizarlo sencillamente en el resto del código.

```
AuthProviderTest.cs
```

```

using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;
    
```

```

namespace BlazorPeliculas.Client.Auth {
    
```

```
public class AuthProviderTest : AuthenticationStateProvider {
    public override async Task<AuthenticationState> GetAuthenticationStateAsync() {
        var anonymous = new ClaimsIdentity();
        var user = new ClaimsIdentity(new List<Claim> {
            new Claim("Key1", "Value1"),
            new Claim("Age", "45"),
            new Claim(ClaimTypes.Name, "Emiliano")
        });
        authenticationType: "test");
        return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(user)));
    }
}
```

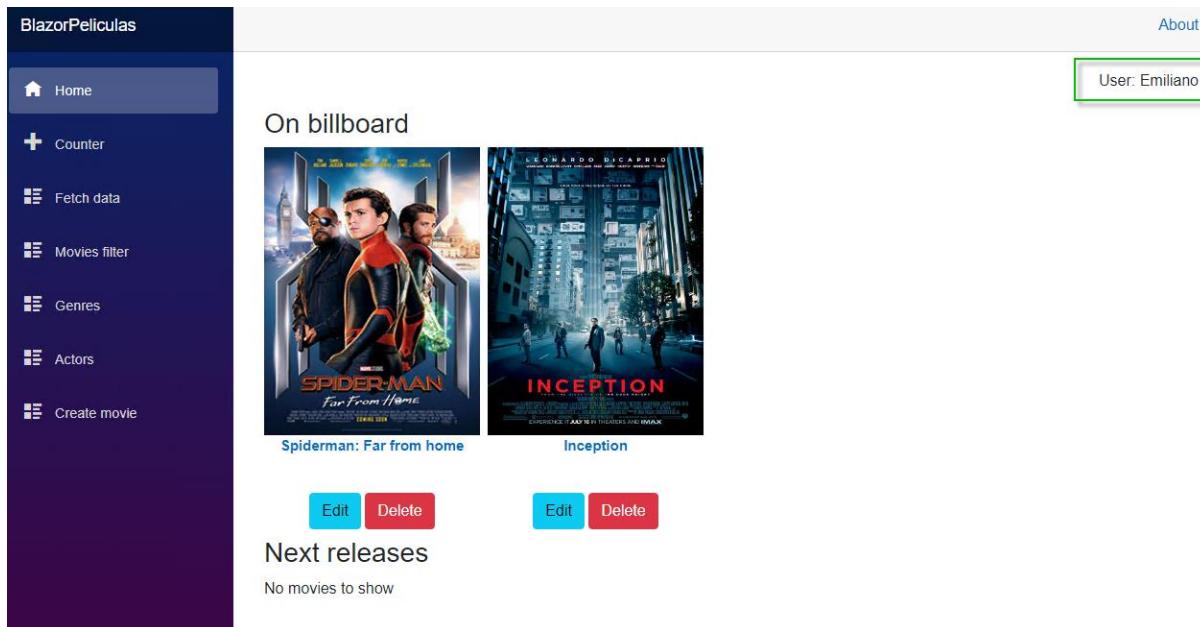
Index.razor

```
@page "/"
@inject IRepository repository

<PageTitle>Blazor Movies</PageTitle>
<AuthorizeView>
    <Authorized>
        <p style="text-align:right;">User: @context.User.Identity!.Name</p>
    </Authorized>
    <NotAuthorized>
        <p>You are NOT authenticated...</p>
    </NotAuthorized>
</AuthorizeView>
<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                
    </Loading>
    <NoRecords>
        <p>No movies to show</p>
    </NoRecords>
    <MoviesList>
    </div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync()
    {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
        OnBoard = responseHTTP.Response!.OnBoard;
        NextReleases = responseHTTP.Response!.NextReleases;
    }
}
    
```



Roles

Como dijéramos antes, estar autenticado no es lo mismo que estar autorizado. Un usuario puede estar autenticado como Emiliano, como en los ejemplos anteriores, pero no tener permiso (no estar autorizado) a ver funcionalidad reservada al rol de administrador. Por ejemplo, con el siguiente cambio en Index.razor, dejaríamos de ver la info que veíamos por carecer del rol admin:

Index.razor

```
@page "/"
@inject IRepository repository

<PageTitle>Blazor Movies</PageTitle>
<AuthorizeView Roles="admin">
    <Authorized>
        <p style="text-align:right;">User: @context.User.Identity!.Name</p>
    </Authorized>
    <NotAuthorized>
        <p>You are NOT authorized...</p>
    </NotAuthorized>
</AuthorizeView>
<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
    }
}
```

```

OnBoard = responseHTTP.Response!.OnBoard;
NextReleases = responseHTTP.Response!.NextReleases;
}
}
    
```

You are NOT authorized...

On billboard

Spiderman: Far from home

Inception

Edit Delete

Edit Delete

Next releases

No movies to show

Para tener visibilidad, debemos agregar el rol a nuestro array de Claims.

AuthProviderTest.cs

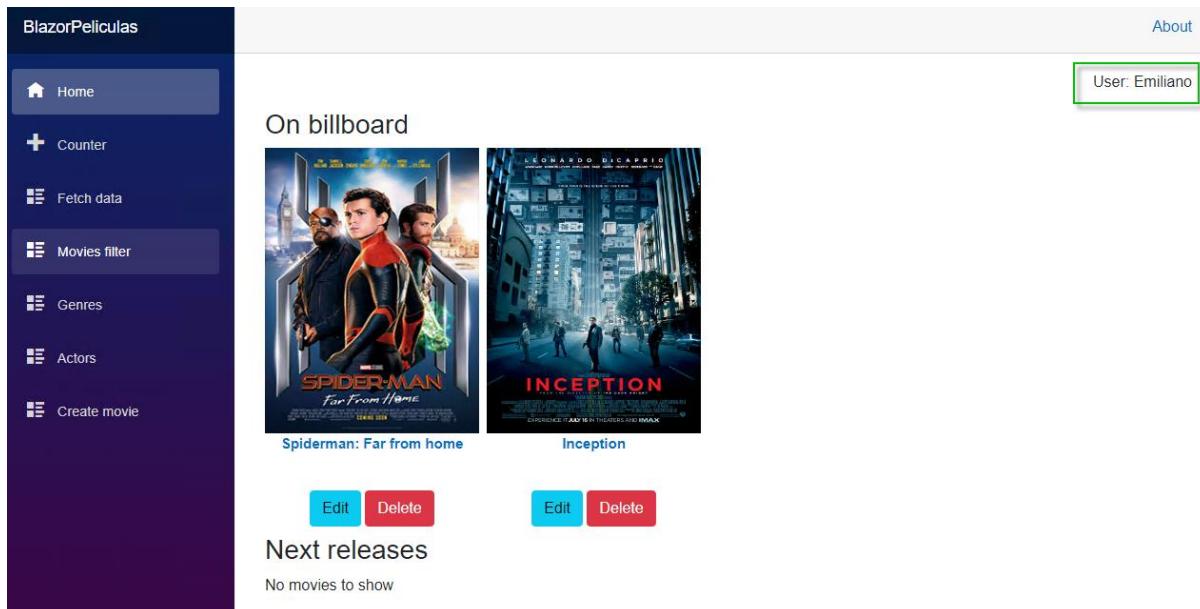
```

using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class AuthProviderTest : AuthenticationStateProvider {
        public override async Task<AuthenticationState> GetAuthenticationStateAsync() {
            var anonymous = new ClaimsIdentity();
            var user = new ClaimsIdentity(new List<Claim> {
                new Claim("Key1", "Value1"),
                new Claim("Age", "45"),
                new Claim(ClaimTypes.Name, "Emiliano",
                new Claim(ClaimTypes.Role, "admin"))
            });
        }
    }
}
    
```

```

        authenticationType: "test");
        return await Task.FromResult(new AuthenticationState(new ClaimsPrincipal(user)));
    }
}
}
    
```



Eliminaremos temporalmente el rol de administrador y aplicaremos los siguientes cambios:

```

NavMenu.razor
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">BlazorPelículas</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="oi oi-plus" aria-hidden="true"></span> Counter
            </NavLink>
        </div>
    </nav>
</div>
    
```

```

</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="fetchdata">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
    </NavLink>
</div>

<div class="nav-item px-3">
    <NavLink class="nav-link" href="movies/filter">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Movies filter
    </NavLink>
</div>

<AuthorizeView Roles="admin">
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="genres">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="actors">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/create">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
        </NavLink>
    </div>
</AuthorizeView>
</nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}

```

MovielItem.razor

```

<div class="me-2 mb-2" style="text-align:center">
    <a href="@urlViewMovie">
        
    </a>

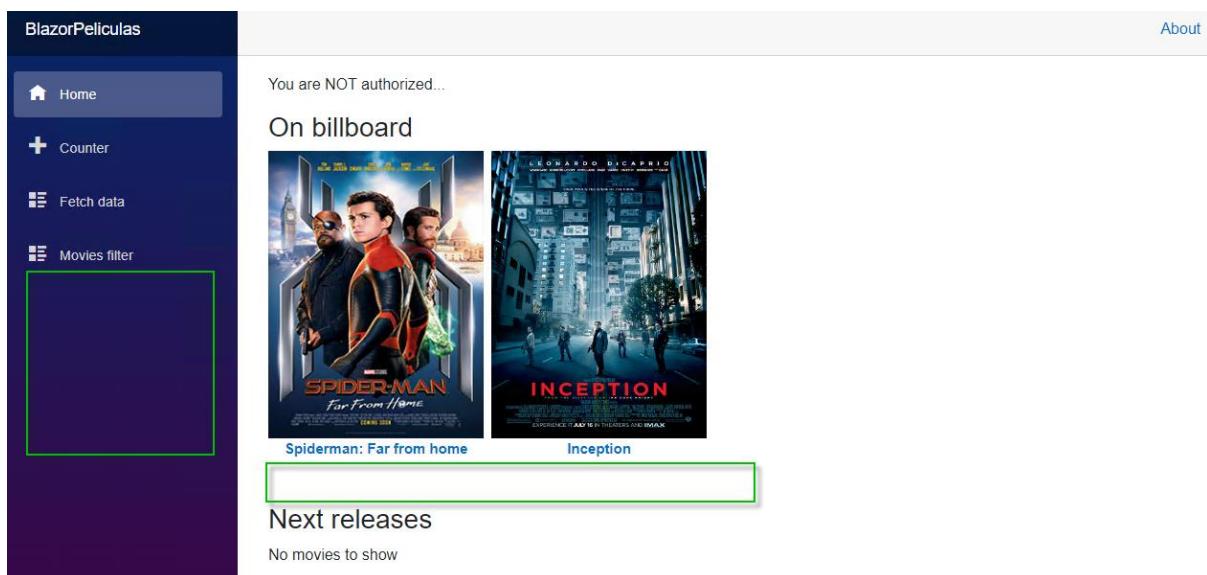
```

```
<p style="max-width:225px;height:44px;font-size:15px;font-weight:bold;">
    <a href="@urlViewMovie" class="text-decoration-none">@Movie.Title</a>
</p>
<AuthorizeView Roles="admin">
    <div>
        <a class="btn btn-info" href="@urlEditMovie">Edit</a>
        <button type="button" class="btn btn-danger"
            @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
    </div>
</AuthorizeView>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }

    private string urlViewMovie = string.Empty;
    private string urlEditMovie = string.Empty;
    protected override void OnInitialized()
    {
        urlViewMovie = $"/movie/{Movie.ID}/{Movie.urlTitle()}";
        urlEditMovie = $"/movie/edit/{Movie.ID}";
    }
}
```

Veremos que no están los botones de **Edit/Delete** en cada película ni los ítems en el panel izquierdo:



Esto sólo oculta los elementos de la pantalla. Sin embargo, habría que asegurar las rutas para que si el usuario se dirigiera a /actors no pudiera ver el listado de actores ni hacer click en los botones de Edit/Delete.

Proteger componentes

Para proteger componentes usamos el atributo **Authorize** del namespace **Microsoft.AspNetCore.Authorization**.

```
GenresList.razor
@page "/genres"
@using Microsoft.AspNetCore.Authorization;
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
<h3>Genres</h3>

<div class="mb-3">
    <a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

<GenericList List="Genres">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
```

```
</thead>
<tbody>
    @foreach(var item in Genres!) {
        <tr>
            <td>
                <a href="/genres/edit/@item.ID" class="btn btn-success">Edit</a>
                <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
            </td>
            <td>@item.Name</td>
        </tr>
    }
</tbody>
</table>
</HasRecordsComplete>
</GenericList>

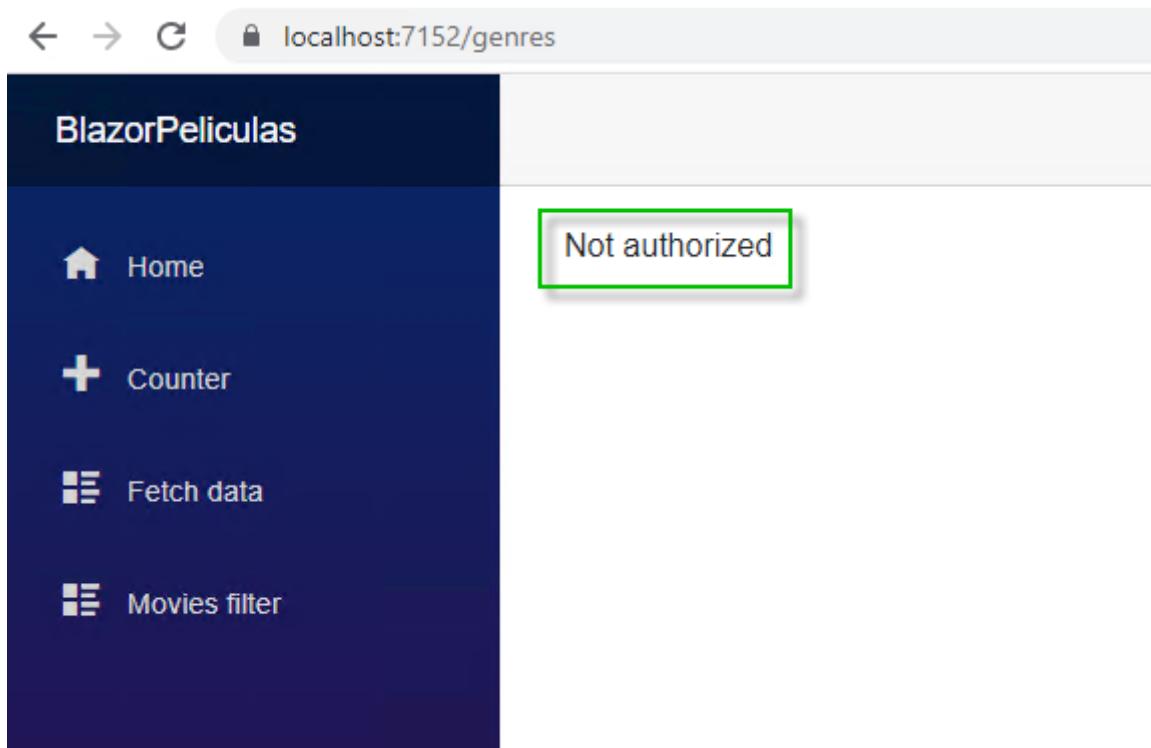
@code {
    public List<Genre>? Genres { get; set; }

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task Load() {
        var responseHTTP = await repository.Get<List<Genre>>("api/genres");
        Genres = responseHTTP.Response;
    }

    public async Task Delete(Genre genre) {
        var responseHTTP = await repository.Delete($"api/genres/{genre.ID}");

        if (responseHTTP.Error) {
            if (responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            await Load();
        }
    }
}
```



Si queremos personalizar ese mensaje, lo hacemos en el [App.razor](#).

App.razor

```
@using Microsoft.AspNetCore.Components.WebAssembly.Services;
@using System.Reflection;
@inject LazyAssemblyLoader lazyLoader

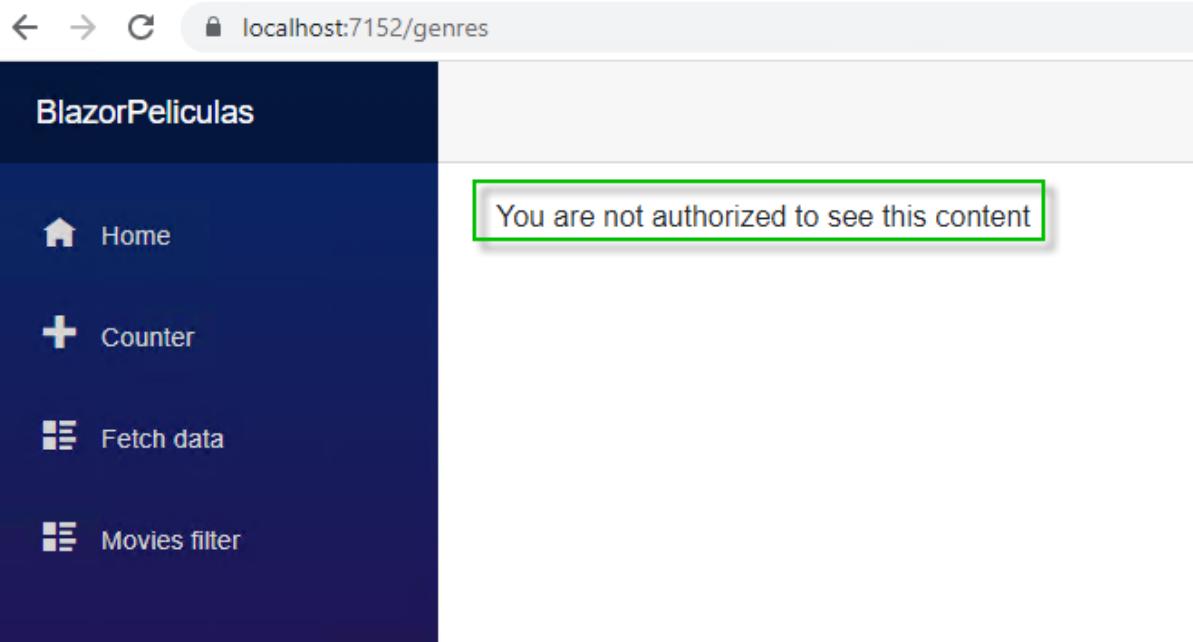
<Router AppAssembly="@typeof(App).Assembly"
OnNavigateAsync="OnNavigateAsync"
AdditionalAssemblies="assemblies">
<Found Context="routeData">
    <AuthorizeRouteView RouteData="@routeData" DefaultLayout="@typeof(MainLayout)">
        <Authorizing>
            <h3>Authorizing user...</h3>
            <LoadingWheel/>
        </Authorizing>
        <NotAuthorized>
            <p>You are not authorized to see this content</p>
        </NotAuthorized>
    </AuthorizeRouteView>
    <FocusOnNavigate RouteData="@routeData" Selector="h1" />
</Found>
<NotFound>
    <CascadingAuthenticationState>
        <PageTitle>Not found</PageTitle>
        <LayoutView Layout="@typeof(MainLayout)">
            <p role="alert">Sorry, there's nothing at this address.</p>
        </LayoutView>
    </CascadingAuthenticationState>
</NotFound>
</Router>
```

```

</LayoutView>
</CascadingAuthenticationState>
</NotFound>
</Router>

@code {
    private List<Assembly> assemblies = new List<Assembly>();
    private async Task OnNavigateAsync(NavigationContext args) {
        if(args.Path.EndsWith("counter")) {
            var loadedAssemblies = await lazyLoader.LoadAssembliesAsync(
                new List<string> { "MathNet.Numerics.dll" }
            );
            assemblies.AddRange(loadedAssemblies);
        }
    }
}

```



Agregaremos estas líneas en los componentes Create*, Edit* y *List.

Usuario autenticado

Haremos un ejemplo con **Counter** para que si el usuario está autenticado sume uno al contador y si no lo está, reste uno.

Para eso, en el código de **Counter**, utilizamos un parámetro de cascada que nos viene configurado por el **AuthorizedRouteView**.

Counter.razor.cs

```

using BlazorPeliculas.Client.Helpers;
using MathNet.Numerics.Statistics;

```

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.JSInterop;

namespace BlazorPeliculas.Client.Pages {
    public partial class Counter {
        [Inject] IJSRuntime js { get; set; } = null!;
        [CascadingParameter] private Task<AuthenticationState> authStateTask { get; set; } = null!;

        private int currentCount = 0;

        public async Task IncrementCount() {
            var arr = new double[]{ 1, 2, 3, 4, 5 };
            var max = arr.Maximum();
            var min = arr.Minimum();

            //await js.InvokeVoidAsync("alert", $"The max value is {max} and the min es {min}");

            var authState = await authStateTask;
            var userIsAuthenticated = authState.User.Identity!.IsAuthenticated;

            if (userIsAuthenticated)
                currentCount++;
            else
                currentCount--;
        }
    }
}
```

Con eso alcanza para saber si el usuario está autenticado o no. Vale aclarar que con **authState.User.Claims** podemos acceder a todos los Claims que tenga el usuario.

Identity

Es una librería que nos permite configurar un sistema de usuarios utilizando .NET. Básicamente con **Identity** tenemos un conjunto de funcionalidades y UI o interfaz de usuario, la cual nos la pone cómoda a la hora de implementar un sistema de usuarios.

Pero no solo eso, sino que también **Identity** tiene puntos de configuración a través de los cuales podemos configurar el sistema de usuarios. Es decir, no es rígido, ya que podemos personalizarlo a nuestro gusto.

Entonces, con este sistema de usuarios vamos a poder permitir que nuestros usuarios se registren, se logueen, que tengan roles, podemos definir permisos a nivel de nuestra webapi para que así solamente determinados usuarios puedan utilizar determinados endpoints o acciones de los controladores.

Vamos al proyecto **Server**, click derecho e instalamos el paquete **NuGet** llamado **Microsoft.AspNetCore.Identity.EntityFrameworkCore**.

En la clase **ApplicationDbContext**, cambiaremos la herencia de **DbContext**, por la de **IdentityDbContext**. La idea de esto es que el Identity de Context contiene un conjunto de entidades, las cuales me van a crear unas tablas, como por ejemplo la tabla de usuarios o la tabla de roles, que son una parte esencial de nuestro sistema de usuarios.

Es muy importante que ya luego de haber hecho esto es asegurarnos de tener el **base.OnModelCreating** en el método **OnModelCreating**.

ApplicationContext.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : IdentityDbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder); //No se puede eliminar esta línea.

            modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });
            modelBuilder.Entity<MovieActor>().HasKey(x => new { x.ActorID, x.MovieID });
        }

        public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase Genre.
        public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
        public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase Movie.
        public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla GenresMovie a partir de la clase GenresMovie.
        public DbSet<MovieActor> MoviesActors => Set<MovieActor>(); //Creamos la tabla MoviesActors a partir de la clase MovieActor.
    }
}
```

Vamos al archivo **Program.cs** del proyecto **Server** y agregamos este servicio nuevo. Utilizaremos la clase **IdentityUser** para identificar un usuario y **IdentityRole** para identificar un rol.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.Identity;
```

```
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews()
    .AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment()) {
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

Ahora crearemos una migración desde el **Package Manager Console** con los comandos:

PMC	EFC cli
Add-Migration IdentityTables	dotnet ef migrations add IdentityTables

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

Package Manager Console Host Version 6.4.0.111

Type 'get-help NuGet' to see all available NuGet commands.

```
PM> Add-Migration IdentityTables
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

Esto nos creará una nueva migración en la que podemos ver que crearía las tablas Roles, Users, etc. La tabla de Roles tiene un campo ConcurrencyStamp es para garantizar que un registro no sea actualizado por 2 personas al mismo tiempo.

20230215163834_IdentityTables.cs

```
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace BlazorPeliculas.Server.Migrations
{
    /// <inheritdoc />
    public partial class IdentityTables : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table => new
                {
                    Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
                    Name = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
                    ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                }
            );
        }

        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "AspNetRoles");
        }
    }
}
```

```
});

migrationBuilder.CreateTable(
    name: "AspNetUsers",
    columns: table => new
    {
        Id = table.Column<string>(type: "nvarchar(450)", nullable: false),
        UserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
        NormalizedUserName = table.Column<string>(type: "nvarchar(256)", maxLength: 256,
nullable: true),
        Email = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable: true),
        NormalizedEmail = table.Column<string>(type: "nvarchar(256)", maxLength: 256, nullable:
true),
        EmailConfirmed = table.Column<bool>(type: "bit", nullable: false),
        PasswordHash = table.Column<string>(type: "nvarchar(max)", nullable: true),
        SecurityStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
        ConcurrencyStamp = table.Column<string>(type: "nvarchar(max)", nullable: true),
        PhoneNumber = table.Column<string>(type: "nvarchar(max)", nullable: true),
        PhoneNumberConfirmed = table.Column<bool>(type: "bit", nullable: false),
        TwoFactorEnabled = table.Column<bool>(type: "bit", nullable: false),
        LockoutEnd = table.Column<DateTimeOffset>(type: "datetimeoffset", nullable: true),
        LockoutEnabled = table.Column<bool>(type: "bit", nullable: false),
        AccessFailedCount = table.Column<int>(type: "int", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_AspNetUsers", x => x.Id);
    });
}

migrationBuilder.CreateTable(
    name: "AspNetRoleClaims",
    columns: table => new
    {
        Id = table.Column<int>(type: "int", nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        RoleId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        ClaimType = table.Column<string>(type: "nvarchar(max)", nullable: true),
        ClaimValue = table.Column<string>(type: "nvarchar(max)", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_AspNetRoleClaims", x => x.Id);
        table.ForeignKey(
            name: "FK_AspNetRoleClaims_AspNetRoles_RoleId",
            column: x => x.RoleId,
            principalTable: "AspNetRoles",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
}
```

```
migrationBuilder.CreateTable(  
    name: "AspNetUserClaims",  
    columns: table => new  
    {  
        Id = table.Column<int>(type: "int", nullable: false)  
            .Annotation("SqlServer:Identity", "1, 1"),  
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false),  
        ClaimType = table.Column<string>(type: "nvarchar(max)", nullable: true),  
        ClaimValue = table.Column<string>(type: "nvarchar(max)", nullable: true)  
    },  
    constraints: table =>  
    {  
        table.PrimaryKey("PK_AspNetUserClaims", x => x.Id);  
        table.ForeignKey(  
            name: "FK_AspNetUserClaims_AspNetUsers_UserId",  
            column: x => x.UserId,  
            principalTable: "AspNetUsers",  
            principalColumn: "Id",  
            onDelete: ReferentialAction.Cascade);  
    });  
  
migrationBuilder.CreateTable(  
    name: "AspNetUserLogins",  
    columns: table => new  
    {  
        LoginProvider = table.Column<string>(type: "nvarchar(450)", nullable: false),  
        ProviderKey = table.Column<string>(type: "nvarchar(450)", nullable: false),  
        ProviderDisplayName = table.Column<string>(type: "nvarchar(max)", nullable: true),  
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false)  
    },  
    constraints: table =>  
    {  
        table.PrimaryKey("PK_AspNetUserLogins", x => new { x.LoginProvider, x.ProviderKey });  
        table.ForeignKey(  
            name: "FK_AspNetUserLogins_AspNetUsers_UserId",  
            column: x => x.UserId,  
            principalTable: "AspNetUsers",  
            principalColumn: "Id",  
            onDelete: ReferentialAction.Cascade);  
    });  
  
migrationBuilder.CreateTable(  
    name: "AspNetUserRoles",  
    columns: table => new  
    {  
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false),  
        RoleId = table.Column<string>(type: "nvarchar(450)", nullable: false)  
    },  
    constraints: table =>  
    {
```

```
table.PrimaryKey("PK_AspNetUserRoles", x => new { x.UserId, x.RoleId });
table.ForeignKey(
    name: "FK_AspNetUserRoles_AspNetRoles_RoleId",
    column: x => x.RoleId,
    principalTable: "AspNetRoles",
    principalColumn: "Id",
    onDelete: ReferentialAction.Cascade);
table.ForeignKey(
    name: "FK_AspNetUserRoles_AspNetUsers_UserId",
    column: x => x.UserId,
    principalTable: "AspNetUsers",
    principalColumn: "Id",
    onDelete: ReferentialAction.Cascade);
});

migrationBuilder.CreateTable(
    name: "AspNetUserTokens",
    columns: table => new
    {
        UserId = table.Column<string>(type: "nvarchar(450)", nullable: false),
        LoginProvider = table.Column<string>(type: "nvarchar(450)", nullable: false),
        Name = table.Column<string>(type: "nvarchar(450)", nullable: false),
        Value = table.Column<string>(type: "nvarchar(max)", nullable: true)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_AspNetUserTokens", x => new { x.UserId, x.LoginProvider, x.Name });
        table.ForeignKey(
            name: "FK_AspNetUserTokens_AspNetUsers_UserId",
            column: x => x.UserId,
            principalTable: "AspNetUsers",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
}

migrationBuilder.CreateIndex(
    name: "IX_AspNetRoleClaims_RoleId",
    table: "AspNetRoleClaims",
    column: "RoleId");
migrationBuilder.CreateIndex(
    name: "RoleNameIndex",
    table: "AspNetRoles",
    column: "Normalized Name",
    unique: true,
    filter: "[Normalized Name] IS NOT NULL");
migrationBuilder.CreateIndex(
    name: "IX_AspNetUserClaims_UserId",
    table: "AspNetUserClaims",
```

```
column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_AspNetUserLogins_UserId",
    table: "AspNetUserLogins",
    column: "UserId");

migrationBuilder.CreateIndex(
    name: "IX_AspNetUserRoles_RoleId",
    table: "AspNetUserRoles",
    column: "RoleId");

migrationBuilder.CreateIndex(
    name: "EmailIndex",
    table: "AspNetUsers",
    column: "NormalizedEmail");

migrationBuilder.CreateIndex(
    name: "UserNameIndex",
    table: "AspNetUsers",
    column: "NormalizedUserName",
    unique: true,
    filter: "[NormalizedUserName] IS NOT NULL");
}

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "AspNetRoleClaims");

    migrationBuilder.DropTable(
        name: "AspNetUserClaims");

    migrationBuilder.DropTable(
        name: "AspNetUserLogins");

    migrationBuilder.DropTable(
        name: "AspNetUserRoles");

    migrationBuilder.DropTable(
        name: "AspNetUserTokens");

    migrationBuilder.DropTable(
        name: "AspNetRoles");

    migrationBuilder.DropTable(
        name: "AspNetUsers");
}
```

Actualizamos la BD con estos comandos:

PMC	EFC cli
update-database	dotnet ef database update

```
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any
licenses to, third-party packages. Some packages may include dependencies which are governed by
additional licenses. Follow the package source (feed) URL to determine any dependencies.
```

Package Manager Console Host Version 6.4.0.111

Type 'get-help NuGet' to see all available NuGet commands.

```
PM> update-database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (17ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT OBJECT_ID(N'[__EFMigrationsHistory']");
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT OBJECT_ID(N'[__EFMigrationsHistory"]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT [MigrationId], [ProductVersion]
    FROM [__EFMigrationsHistory]
    ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20230215163834_IdentityTables'.
    Applying migration '20230215163834_IdentityTables'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE TABLE [AspNetRoles] (
        [Id] nvarchar(450) NOT NULL,
        [Name] nvarchar(256) NULL,
        [Normalized Name] nvarchar(256) NULL,
        [ConcurrencyStamp] nvarchar(max) NULL,
        CONSTRAINT [PK_AspNetRoles] PRIMARY KEY ([Id])
    );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE TABLE [AspNetUsers] (
        [Id] nvarchar(450) NOT NULL,
        [UserName] nvarchar(256) NULL,
        [NormalizedUserName] nvarchar(256) NULL,
        [Email] nvarchar(256) NULL,
        [NormalizedEmail] nvarchar(256) NULL,
        [EmailConfirmed] bit NOT NULL,
        [PasswordHash] nvarchar(max) NULL,
        [SecurityStamp] nvarchar(max) NULL,
        [ConcurrencyStamp] nvarchar(max) NULL,
        [PhoneNumber] nvarchar(max) NULL,
        [PhoneNumberConfirmed] bit NOT NULL,
        [TwoFactorEnabled] bit NOT NULL,
        [LockoutEnd] datetimeoffset NULL,
        [LockoutEnabled] bit NOT NULL,
        [AccessFailedCount] int NOT NULL,
        CONSTRAINT [PK_AspNetUsers] PRIMARY KEY ([Id])
    );
```

```

Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [AspNetRoleClaims] (
    [Id] int NOT NULL IDENTITY,
    [RoleId] nvarchar(450) NOT NULL,
    [ClaimType] nvarchar(max) NULL,
    [ClaimValue] nvarchar(max) NULL,
    CONSTRAINT [PK_AspNetRoleClaims] PRIMARY KEY ([Id]),
    CONSTRAINT [FK_AspNetRoleClaims_AspNetRoles_RoleId] FOREIGN KEY ([RoleId]) REFERENCES
[AspNetRoles] ([Id]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (3ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [AspNetUserClaims] (
    [Id] int NOT NULL IDENTITY,
    [UserId] nvarchar(450) NOT NULL,
    [ClaimType] nvarchar(max) NULL,
    [ClaimValue] nvarchar(max) NULL,
    CONSTRAINT [PK_AspNetUserClaims] PRIMARY KEY ([Id]),
    CONSTRAINT [FK_AspNetUserClaims_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES
[AspNetUsers] ([Id]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [AspNetUserLogins] (
    [LoginProvider] nvarchar(450) NOT NULL,
    [ProviderKey] nvarchar(450) NOT NULL,
    [ProviderDisplayName] nvarchar(max) NULL,
    [UserId] nvarchar(450) NOT NULL,
    CONSTRAINT [PK_AspNetUserLogins] PRIMARY KEY ([LoginProvider], [ProviderKey]),
    CONSTRAINT [FK_AspNetUserLogins_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES
[AspNetUsers] ([Id]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [AspNetUserRoles] (
    [UserId] nvarchar(450) NOT NULL,
    [RoleId] nvarchar(450) NOT NULL,
    CONSTRAINT [PK_AspNetUserRoles] PRIMARY KEY ([UserId], [RoleId]),
    CONSTRAINT [FK_AspNetUserRoles_AspNetRoles_RoleId] FOREIGN KEY ([RoleId]) REFERENCES
[AspNetRoles] ([Id]) ON DELETE CASCADE,
    CONSTRAINT [FK_AspNetUserRoles_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES
[AspNetUsers] ([Id]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [AspNetUserTokens] (
    [UserId] nvarchar(450) NOT NULL,
    [LoginProvider] nvarchar(450) NOT NULL,
    [Name] nvarchar(450) NOT NULL,
    [Value] nvarchar(max) NULL,
    CONSTRAINT [PK_AspNetUserTokens] PRIMARY KEY ([UserId], [LoginProvider], [Name]),
    CONSTRAINT [FK_AspNetUserTokens_AspNetUsers_UserId] FOREIGN KEY ([UserId]) REFERENCES
[AspNetUsers] ([Id]) ON DELETE CASCADE
);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_AspNetRoleClaims_RoleId] ON [AspNetRoleClaims] ([RoleId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE UNIQUE INDEX [RoleNameIndex] ON [AspNetRoles] ([Normalized Name]) WHERE [Normalized Name]
IS NOT NULL;
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_AspNetUserClaims_UserId] ON [AspNetUserClaims] ([UserId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE INDEX [IX_AspNetUserLogins_UserId] ON [AspNetUserLogins] ([UserId]);

```

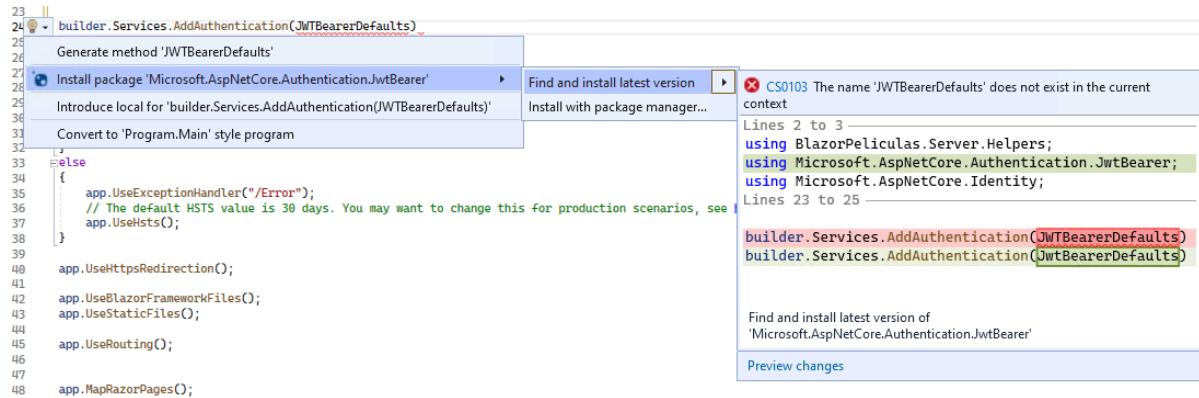
```

Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE INDEX [IX_AspNetUserRoles_RoleId] ON [AspNetUserRoles] ([RoleId]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE INDEX [EmailIndex] ON [AspNetUsers] ([NormalizedEmail]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE UNIQUE INDEX [UserNameIndex] ON [AspNetUsers] ([NormalizedUserName]) WHERE
[NormalizedUserName] IS NOT NULL;
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
    VALUES (N'20230215163834_IdentityTables', N'7.0.2');
Done.
PM>

```

Ahor utilizaremos JSON webtokens para autorizar los usuarios en nuestra api. En nuestros JWT tendremos los claims del usuario de manera cifrada. Para asegurar que el JWT no fue modificado, la info se firma con una llave secreta.

Para conseguir esto agregamos el servicio de autenticación en **Program.cs** utilizando un paquete de **NuGet**. Sencillamente escribimos el parámetro **JwtBearerDefaults** y al presionar **Ctrl+**, nos ofrece buscar e instalar la última versión:



Esto habrá instalado la última versión del paquete correspondiente. En este caso, la v7.0.3:

```

BlazorPeliculas.Server*  Program.cs*  Index.razor  Counter.razor.cs  AuthProviderTest.cs
1  <Project Sdk="Microsoft.NET.Sdk.Web">
2
3      <PropertyGroup>
4          <TargetFramework>net7.0</TargetFramework>
5          <Nullable>enable</Nullable>
6          <ImplicitUsings>enable</ImplicitUsings>
7      </PropertyGroup>
8
9      <ItemGroup>
10         <PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="12.0.0" />
11         <PackageReference Include="Azure.Storage.Blobs" Version="12.14.1" />
12         <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="7.0.3" /> (highlighted)
13         <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.Server" Version="7.0.2" />
14         <PackageReference Include="Microsoft.AspNetCore.Identity.EntityFrameworkCore" Version="7.0.2" />
15         <PackageReference Include="Microsoft.EntityFrameworkCore.SqlServer" Version="7.0.2" />
16         <PackageReference Include="Microsoft.EntityFrameworkCore.Tools" Version="7.0.2" />
17             <PrivateAssets>all</PrivateAssets>
18             <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
19         </PackageReference>
20     </ItemGroup>
21
22     <ItemGroup>
23         <ProjectReference Include="..\Client\BlazorPeliculas.Client.csproj" />
24         <ProjectReference Include="..\Shared\BlazorPeliculas.Shared.csproj" />
25     </ItemGroup>
26
27     <ItemGroup>
28         <Folder Include="wwwroot\" />
29     </ItemGroup>
30
31
32 </Project>
33

```

En la configuración del servicio, aclaramos las opciones que queremos utilizar con nuestros JWT: le indicamos que no queremos validar el issuer (quien lo emite) ni la audiencia (receptores) pero **sí** queremos validar el lifetime (si es válido por una hora, no se puede usar luego de ese tiempo) y el IssuerSigningKey (la firma del emisor, la clave secreta que me garantiza que el JWT no fue alterado). Para ello, le indicamos cuál es nuestra clave (que deberá ser secreta para que el JWT sea seguro) que configuraremos en el archivo **appsettings.Development.json**.

```

appsettings.Development.json
{
    "ConnectionStrings": {
        "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=BlazorPeliculas;Trusted_Connection=True",
        "AzureStorage": "blablabla"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "jwtkey": "utLEZPsC96sxppPqo5KtvI8Gm1VpoiGdVfdOZvcdpVgBcMbNLAxI7M9bsRwW1PUBY05vBh5laA5bXw9MXyJNUg3SOPDZWpCEh1587RZ2OOomLGTK0kPKLZJtFxR7k1"
}

```

El **ClockSkew** es para que no tenga errores con el tema del tiempo de vida de nuestro JWT.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews()
    .AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
        options.TokenValidationParameters = new TokenValidationParameters {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(builder.Configuration["jwtkey"]!)),
            ClockSkew = TimeSpan.Zero
        });
}

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
```

```
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

Construyendo JWT

Crearemos un controlador que permitirá a los usuarios registrarse y loguearse. Creamos, entonces, la clase **AccountsController** en la carpeta **Controllers** del proyecto **Server** y la hacemos heredar de **ControllerBase**. Le agregamos los atributos de **ApiController** y **Route**.

En el constructor inyectamos el **UserManager** que nos permitirá trabajar con el sistema de usuarios y le indicamos que la clase es **IdentityUser** (clase que representa a un usuario en mi sistema), el **SignInManager** nos permitirá autenticar a los usuarios y el **IConfiguration** para poder tomar las configuraciones del sistema del json.

Antes de seguir, crearemos un DTO para manejar la info del usuario. Creamos la clase **UserInfoDTO.cs** en la carpeta **DTOs** del proyecto **Server**.

```
UserInfoDTO.cs
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
```

```
public class UserInfoDTO {  
    [EmailAddress]  
    public string Email { get; set; } = null!;  
    public string Password { get; set; } = null!;  
}
```

También uno llamado **UserTokenDTO** para manejar la información del token:

UserTokenDTO.cs

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace BlazorPeliculas.Shared.DTOs {  
    public class UserTokenDTO {  
        public string Token { get; set; } = null!;  
        public DateTime Expiration { get; set; }  
    }  
}
```

El controlador de cuentas, entonces queda así:

AccountsController.cs

```
using BlazorPeliculas.Shared.DTOs;  
using Microsoft.AspNetCore.Identity;  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.IdentityModel.Tokens;  
using System.IdentityModel.Tokens.Jwt;  
using System.Security.Claims;  
using System.Text;  
  
namespace BlazorPeliculas.Server.Controllers {  
    [ApiController, Route("api/accounts")]  
    public class AccountsController : ControllerBase {  
        private readonly UserManager<IdentityUser> userManager;  
        private readonly SignInManager<IdentityUser> signInManager;  
        private readonly IConfiguration configuration;  
  
        public AccountsController(UserManager<IdentityUser> userManager,  
            SignInManager<IdentityUser> signInManager,  
            IConfiguration configuration) {  
            this.userManager = userManager;  
            this.signInManager = signInManager;  
            this.configuration = configuration;  
        }  
    }  
}
```

```
[HttpPost("create")]
public async Task<ActionResult<UserTokenDTO>> CreateUser([FromBody] UserInfoDTO model) {
    var user = new IdentityUser { UserName = model.Email, Email = model.Email };
    var result = await userManager.CreateAsync(user, model.Password);

    if(result.Succeeded) {
        return BuildToken(model);
    }
    else
        return BadRequest(result.Errors.First());
}

[HttpPost("login")]
public async Task<ActionResult<UserTokenDTO>> Login([FromBody] UserInfoDTO model) {
    //isPersistent: lo guarda en cookies
    //lockoutOnFailure: se lockea el usuario después varios intentos incorrectos.
    var result = await signInManager.PasswordSignInAsync(model.Email, model.Password,
isPersistent: false, lockoutOnFailure: false);

    if(result.Succeeded)
        return BuildToken(model);
    else
        return BadRequest("User/Password incorrect");
}

private UserTokenDTO BuildToken(UserInfoDTO userInfo) {
    var claims = new List<Claim>() {
        //Esta info es accesible desde el frontend de Blazor. NO VA NINGÚN DATO SENSIBLE (clave,
tarjetas de crédito, etc)
        new Claim(ClaimTypes.Name, userInfo.Email),
        new Claim("miValor", "Lo qu necesite...")
    };

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["jwtkey"]));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var expiration = DateTime.UtcNow.AddHours(1);

    var token = new JwtSecurityToken(
        issuer: null,
        audience: null,
        claims: claims,
        expires: expiration,
        signingCredentials: creds
    );

    return new UserTokenDTO {
        Token = new JwtSecurityTokenHandler().WriteToken(token),
        Expiration = expiration
    };
}
```

```
    };
}
}
```

Para que nuestra app utilice autenticación/autorización es necesario indicarlo en [Program.cs](#).

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews()
    .AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationContext>(options =>
options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>()
    .AddDefaultTokenProviders();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
{
    options.TokenValidationParameters = new TokenValidationParameters {
        ValidateIssuer = false,
        ValidateAudience = false,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(
            Encoding.UTF8.GetBytes(builder.Configuration["jwtkey"]!)),
        ClockSkew = TimeSpan.Zero
    }
});
```

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

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapRazorPages();
app.MapControllers();
app.MapFallbackToFile("index.html");

app.Run();
```

Para probar la segurización de las api, agregaremos un atributo que oblique al usuario a tener un JWT para poder ser consumido:

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
```

```
private readonly IFileSaver fileSaver;
private readonly IMapper mapper;
private readonly string container = "movies";

public MoviesController(ApplicationDbContext context,
    IFileSaver fileSaver,
    IMapper mapper) {
    this.context = context;
    this.fileSaver = fileSaver;
    this.mapper = mapper;
}

[HttpGet, Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task<ActionResult<HomePageDTO>> Get() {
    var limit = 6;
    var onBoardMovies = await context.Movies
        .Where(movie => movie.OnBillboard)
        .Take(limit)
        .OrderByDescending(movie => movie.ReleaseDate)
        .ToListAsync();
    var today = DateTime.Today;
    var nextReleases = await context.Movies
        .Where(movie => movie.ReleaseDate > today)
        .Take(limit)
        .OrderBy(movie => movie.ReleaseDate)
        .ToListAsync();

    var result = new HomePageDTO {
        OnBoard = onBoardMovies,
        NextReleases = nextReleases
    };
    return result;
}

[HttpGet("{id:int}")]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }

    //TODO: Sistema de votación.
```

```
var votesMedia = 4;
var userVote = 5;

var model = new MovieViewDTO();
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("filter")]
public async Task<ActionResult<List<Movie>>> Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);

    if(model.Releases) {
        var today = DateTime.Today;

        queryableMovies = queryableMovies
            .Where(x => x.ReleaseDate >= today);
    }

    if (model.GenreID != 0)
        queryableMovies = queryableMovies
            .Where(x => x.GenresMovie
                .Select(y => y.GenreID)
                .Contains(model.GenreID));
    }

    //TODO: Implementar votación

    await HttpContext.InserPaginationParametersInResponse(queryableMovies,
model.RecordCount);
```

```
//Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, ".jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}
```

```
[HttpPost]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if (movie is null)
        return NotFound();

    context.Remove(movie); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if (!string.IsNullOrWhiteSpace(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

    return NoContent();     //Todo se hizo correctamente
}
```

Y para que el índice no se cuelgue (como no tenemos try/catch), listaremos el error en la consola si no pudiera traer el listado:

Index.razor

```
@page "/"
@inject IRepository repository
```

```
<PageTitle>Blazor Movies</PageTitle>
```

```

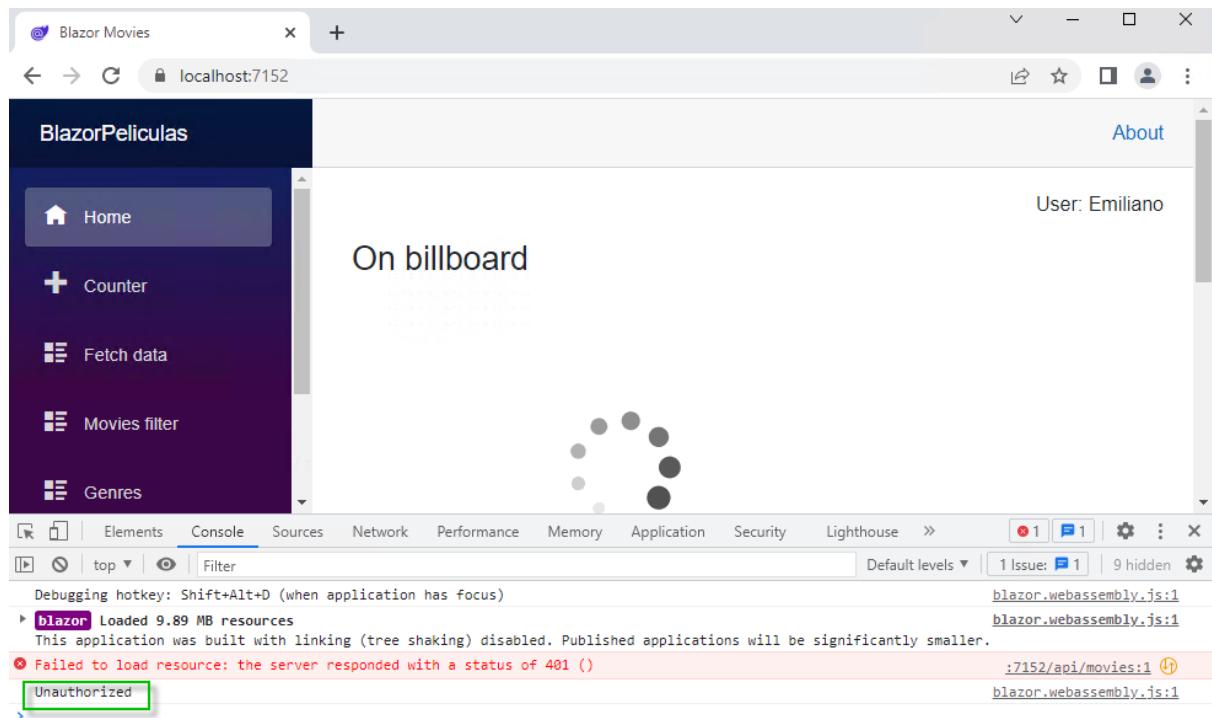
<AuthorizeView Roles="admin">
    <Authorized>
        <p style="text-align:right;">User: @context.User.Identity!.Name</p>
    </Authorized>
</AuthorizeView>
<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
        if(responseHTTP httpResponseMessage.IsSuccessStatusCode) {
            OnBoard = responseHTTP.Response!.OnBoard;
            NextReleases = responseHTTP.Response!.NextReleases;
        }
        else
            Console.WriteLine(responseHTTP httpResponseMessage.StatusCode);
    }
}

```

Al correr la app, vemos que no trae nunca el listado y que escribe en la consola que no se está autorizado:



Vamos a crear un nuevo **AuthenticationStateProvider** para poder tomar los claims del usuario a partir del JWT que recibimos del web API. Para eso crearemos la clase **JWTAuthProvider.cs** en la carpeta **Auth** del proyecto **Client** y heredaremos de **AuthenticationStateProvider**.

Para que el usuario no tenga que loguearse cada vez que sale, usaremos **LocalStorage** para guardar pequeñas piezas de información en la máquina del usuario. Agregamos los métodos para grabar, leer y borrar del LocalStorage usando las funciones de JS. Para ello, vamos a la carpeta **Helpers** y los agregamos en esta clase estática.

IJSRuntimeExtensionMethods.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.JSInterop;
using System.Runtime.CompilerServices;

namespace BlazorPeliculas.Client.Helpers {
    public static class IJSRuntimeExtensionMethods {
        public static async ValueTask<bool> Confirm(this IJSRuntime js, string message) {
            await js.InvokeVoidAsync("console.log", $"Asking: «{message}»");
            return await js.InvokeAsync<bool>("confirm", message);
        }
        public static ValueTask<object> SetInLocalStorage(this IJSRuntime js,
            string key, string value) {
    
```

```
        return js.InvokeAsync<object>("localStorage.setItem", key, value);
    }

    public static ValueTask<object> GetFromLocalStorage(this IJSRuntime js,
        string key) {
        return js.InvokeAsync<object>("localStorage.getItem", key);
    }

    public static ValueTask<object> RemoveFromLocalStorage(this IJSRuntime js,
        string key) {
        return js.InvokeAsync<object>("localStorage.removeItem", key);
    }
}
```

Recordemos que usaremos el JWT para autenticar en el web API, lo que quiere decir que cada vez que utilicemos el HTTP client queremos enviar a través de este el JWT. Lo inyectamos, entonces, en el constructor para poder utilizarlo.

Usaremos el HttpClient que teníamos configurado como Singleton. Es decir, que se utilizará la misma instancia **SIEMPRE**.

Queremos poder obtener los claims del JWT. Para conseguir esto, utilizaremos un paquete NuGet de Microsoft llamado **Microsoft.System.IdentityModel.Tokens.Jwt**.

JWTAuthProvider.cs

```
using BlazorPeliculas.Client.Helpers;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.JSInterop;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http.Headers;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class JWTAuthProvider : AuthenticationStateProvider {
        private readonly IJSRuntime js;
        private readonly HttpClient httpClient;

        public JWTAuthProvider(IJSRuntime js, HttpClient httpClient) {
            this.js = js;
            this.httpClient = httpClient;
        }

        public static readonly string TOKENKEY = "TOKENKEY";
        private AuthenticationState anonymous = new AuthenticationState(new ClaimsPrincipal(new
ClaimsIdentity()));
        public async override Task<AuthenticationState> GetAuthenticationStateAsync() {
            var token = await js.GetFromLocalStorage(TOKENKEY);
```

```
if(token is null token is null) {
    //Es un usuario anónimo.
    return anonymous;
}

return ConstructAuthenticationState(token.ToString()!);
}

private AuthenticationState ConstructAuthenticationState(string token) {
    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
token);
    var claims = ParseClaimsOutOfJWT(token);
    return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));
}

private IEnumerable<Claim> ParseClaimsOutOfJWT(string token) {
    var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
    var deserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

    return deserializedToken.Claims;
}
}
```

Ahora, tenemos que cambiar el servicio de autenticación en [Program.cs](#) para que deje de usar el de prueba y comience a utilizar el real:

Program.cs

```
using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Auth;
using BlazorPeliculas.Client.Repositories;
using CurrieTechnologies.Razor.SweetAlert2;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new
Uri(builder.HostEnvironment.BaseAddress) });
configureServices(builder.Services);
await builder.Build().RunAsync();

void configureServices(IServiceCollection services) {
    services.AddScoped< IRepository, Repository>(); /* Inyectar un IRepository.
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
    clase Repository. */
```

```
services.AddSweetAlert2();
services.AddAuthorizationCore();
services.AddScoped<AuthenticationStateProvider, JWTAuthProvider>();
}
```

Sin embargo, para esto faltaría un servicio para realizar el login/logout. Con lo cual, agregamos una nueva clase del tipo **Interfaz** en la carpeta **Auth** del proyecto **Client**.

ILoginService.cs

```
namespace BlazorPeliculas.Client.Auth {
    public interface ILoginService {
        Task Login(string token);
        Task Logout();
    }
}
```

Agregamos en JWTAuthProvider la herencia de ILoginService y presionamos **Ctrl+** para solicitar que implemente las funciones de la interfaz.

JWTAuthProvider.cs

```
using BlazorPeliculas.Client.Helpers;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.JSInterop;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http.Headers;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class JWTAuthProvider : AuthenticationStateProvider, ILoginService {
        private readonly IJSRuntime js;
        private readonly HttpClient httpClient;

        public JWTAuthProvider(IJSRuntime js, HttpClient httpClient) {
            this.js = js;
            this.httpClient = httpClient;
        }

        public static readonly string TOKENKEY = "TOKENKEY";
        private AuthenticationState anonymous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity()));

        public async override Task<AuthenticationState> GetAuthenticationStateAsync() {
            var token = await js.GetFromLocalStorage(TOKENKEY);

            if(token is null) {
                //Es un usuario anónimo.
                return anonymous;
            }
        }
    }
}
```

```

        }

        return ConstructAuthenticationState(token.ToString()!);
    }

    private AuthenticationState ConstructAuthenticationState(string token) {
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
token);
        var claims = ParseClaimsOutOfJWT(token);
        return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));
    }

    private IEnumerable<Claim> ParseClaimsOutOfJWT(string token) {
        var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
        var deserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

        return deserializedToken.Claims;
    }

    public async Task Login(string token) {
        await js.SetInLocalStorage(TOKENKEY, token);
        var authState = ConstructAuthenticationState(token);
        NotifyAuthenticationStateChanged(Task.FromResult(authState)); //Le notifico a Blazor que
cambió el estado.

    }

    public async Task Logout() {
        await js.RemoveFromLocalStorage(TOKENKEY);
        httpClient.DefaultRequestHeaders.Authorization = null;
        NotifyAuthenticationStateChanged(Task.FromResult(anonymous)); //Le notifico a Blazor que
cambió el estado.
    }
}

```

Ahora necesitamos registrar este servicio en la clase **Program.cs** y aquí tenemos un pequeño problema: estamos utilizando el **JWTAuthProvider** tanto para el **AuthenticationStateProvider** como para el **ILoginService**.

Para resolver esto, primero registramos en nuestro sistema de inyección de dependencias al **JWTAuthProvider**. Luego de registrarlo una vez, podemos reutilizarlo para agregar los 2 servicios.

Program.cs

```

using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Auth;
using BlazorPeliculas.Client.Repositories;
using CurrieTechnologies.Razor.SweetAlert2;

```

```
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Microsoft.Extensions.DependencyInjection;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });
configureServices(builder.Services);
await builder.Build().RunAsync();

void configureServices(IServiceCollection services) {
    services.AddScoped< IRepository, Repository>(); /* Inyectar un IRepository.
        Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
        clase Repository. */
    services.AddSweetAlert2();
    services.AddAuthorizationCore();

    services.AddScoped< JWTAuthProvider >;
    services.AddScoped< AuthenticationStateProvider, JWTAuthProvider >(provider =>
        provider.GetRequiredService< JWTAuthProvider >);

    services.AddScoped< ILoginService, JWTAuthProvider >(provider =>
        provider.GetRequiredService< JWTAuthProvider >);
}
```

Crearemos un componente para los links de registro, login y logout. Para ello, creamos el archivo **AuthLinks.razor** en la carpeta **Shared** del proyecto **Client**.

AuthLinks.cs

```
<AuthorizeView>
    <Authorized>
        <span>Hola, @context.User.Identity.Name!</span>
        <a href="Logout" class="náv-link btn btn-link">Log out</a>
    </Authorized>
    <NotAuthorized>
        <a href="Register">Register</a>
        <a href="Login" class="náv-link btn btn-link">Login</a>
    </NotAuthorized>
</AuthorizeView>
```

Y agregamos el nuevo componente en el **MainLayout.razor**:

```
MainLayout.razor
@inherits LayoutComponentBase

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <AuthLinks />
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>
```

Creamos el componente **Register.razor**, **Login.razor** y **Logout.razor** en la carpeta **Auth** del proyecto **Client**.

```
Register.razor
@page "/Register"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal
@inject ILoginService loginService

<h3>Register</h3>
<EditForm Model="userInfo" OnValidSubmit="CreateUser">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Email:</label>
        <div>
            <InputText class="form-control" @bind-Value="userInfo.Email" />
            <ValidationMessage For="@(() => userInfo.Email)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Password:</label>
        <div>
            <InputText type="text" class="form-control" @bind-Value="userInfo.Password" />
            <ValidationMessage For="@(() => userInfo.Password)" />
        </div>
    </div>
```

```
</div>

<button type="submit" class="btn btn-primary">Register</button>
</EditForm>

@code {
    private UserInfoDTO userInfo = new UserInfoDTO();

    private async Task CreateUser() {
        var responseHTTP = await repository.Post<UserInfoDTO, UserTokenDTO>("api/accounts/create",
userInfo);

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            await loginService.Login(responseHTTP.Response!.Token);
            navManager.NavigateTo("");
        }
    }
}
```

Login.razor

```
@page "/Login"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
@inject ILoginService loginService

<h3>Login</h3>
<EditForm Model="userInfo" OnValidSubmit="LoginUser">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Email:</label>
        <div>
            <InputText class="form-control" @bind-Value="userInfo.Email" />
            <ValidationMessage For="@(() => userInfo.Email)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Password:</label>
        <div>
            <InputText type="password" class="form-control" @bind-Value="userInfo.Password" />
            <ValidationMessage For="@(() => userInfo.Password)" />
        </div>
    </div>
</EditForm>
```

```

<button type="submit" class="btn btn-primary">Login</button>
</EditForm>

@code {
    private UserInfoDTO userInfo = new UserInfoDTO();

    private async Task LoginUser() {
        var responseHTTP = await repository.Post<UserInfoDTO, UserTokenDTO>("api/accounts/login",
userInfo);

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            await loginService.Login(responseHTTP.Response!.Token);
            navManager.NavigateTo("");
        }
    }
}
    
```

```

Logout.razor
@page "/Logout"
@inject ILoginService loginService
@inject NavigationManager navManager

<p>Logging out...</p>

@code {
    protected async override Task OnInitializedAsync() {
        await loginService.Logout();
        navManager.NavigateTo("/");
    }
}
    
```

Antes de probar el registro, vemos que la tabla de Users está vacía:

dbo.AspNetUsers [Data]															Logout.razor	Login.razor	Register.razor	
	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencySt...	PhoneNumber	PhoneNumber...	TwoFactorEna...	LockoutEnd	LockoutEnabled	AccessFailedCount			
	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL			

Probamos el registro y vemos que ya aparecen los links a la derecha. **Identity** tiene algunas reglas: minúsculas, mayúsculas, números y carácter no alfanumérico.

localhost:7152/Register

BlazorPelículas

- Home
- Counter
- Fetch data
- Movies filter

Register

Email:

Password:

Register

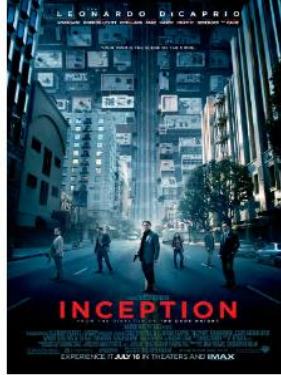
Al presionar Register:

BlazorPelículas

- Home
- Counter
- Fetch data
- Movies filter

Hola, emiliano@blazor.com! [Log out](#) [About](#)

On billboard

[Spiderman: Far from home](#) [Inception](#)

Next releases

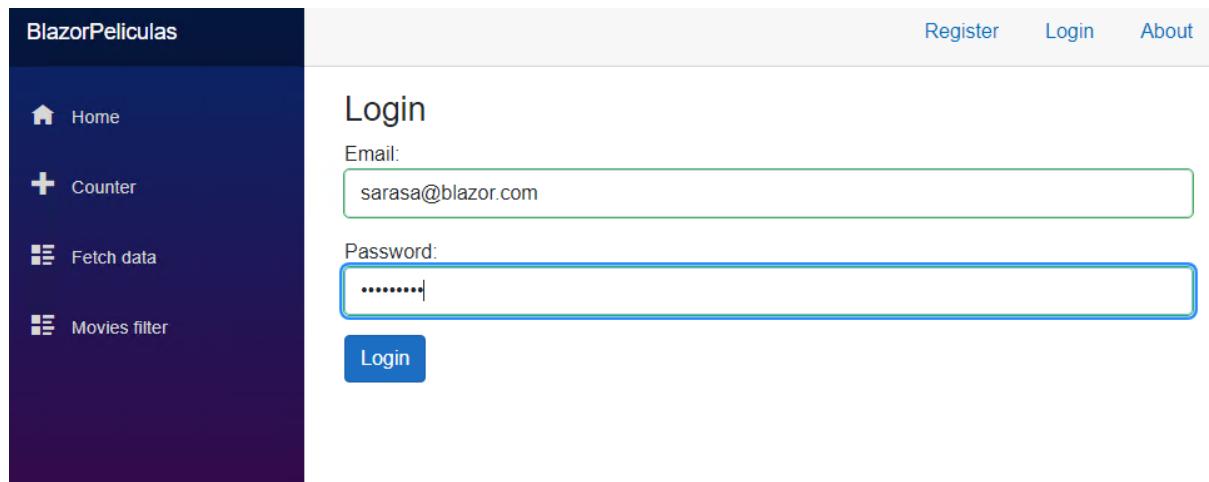
No movies to show

Vemos que nos registró y logueó: Se lee el usuario logueado a la derecha y el link de logout. Así también, vemos que el usuario ya existe en la tabla:

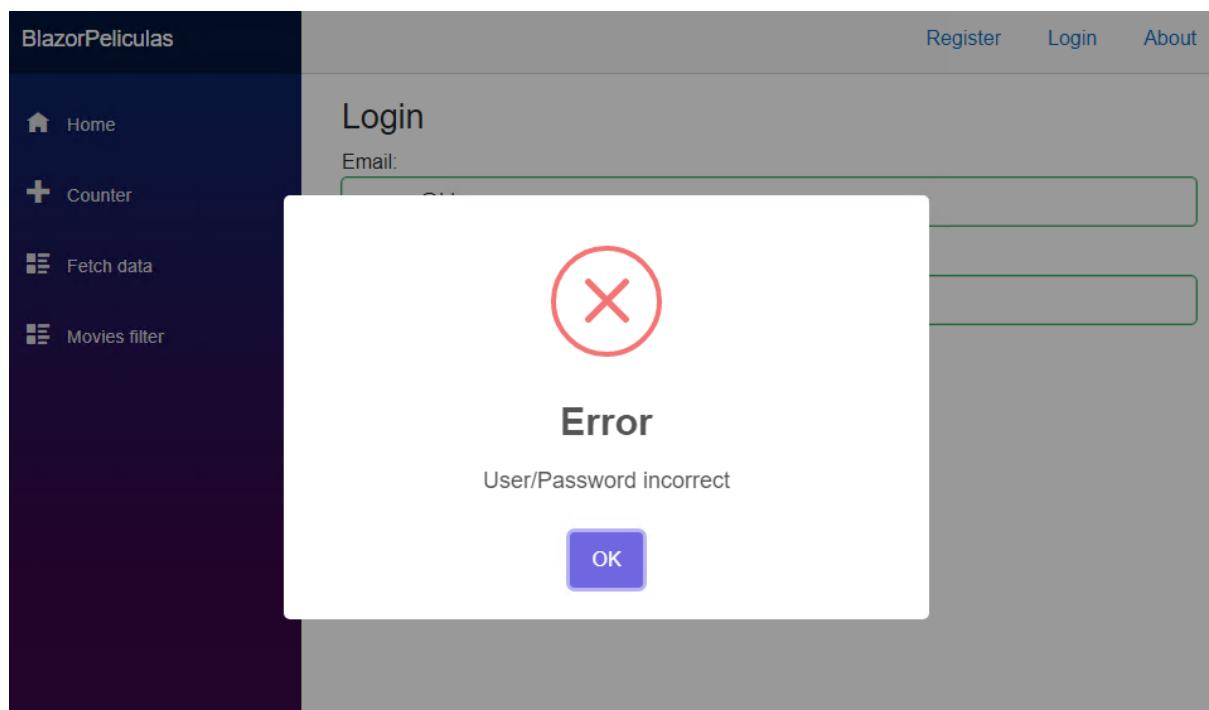
dbo.AspNetUsers [Data]															
	Id	UserName	NormalizedUserName	Email	NormalizedEmail	EmailConfirmed	PasswordHash	SecurityStamp	ConcurrencySt...	PhoneNumber	PhoneNumber...	TwoFactorEna...	LockoutEnd	LockoutEnabled	AccessFailedCount
▶	24255	emiliano...	EMILIANO@BLAZOR...	emil...	EMILIANO@BL...	False	AQAAAAIAAYA...	KVWPZTSOW...	318aa7b1-39fd...	NULL	False	False	NULL	True	0
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

a

Intentaremos el login con un usuario cualquiera:



Al presionar Login:



Atributo Authorize

Anteriormente utilizamos el atributo:

```
Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)
```

Para impedir que se pudiera consumir el Get de películas (un ejemplo, ya que no es útil en la vida real). Algo más práctico es seguirizar todas las apis y permitir el acceso anónimo a las que no lo necesiten. Por ejemplo, un usuario no logueado debería poder ver el listado, la info de una película o filtrar. Esto se logra poniendo

al restricción arriba de todo y permitir el acceso anónimo con el atributo **AllowAnonymous** en las apis que querramos:

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;
```

```
namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet, AllowAnonymous]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
                .Take(limit)
                .OrderBy(movie => movie.ReleaseDate)
                .ToListAsync();

            var result = new HomePageDTO {
                OnBoard = onBoardMovies,
                NextReleases = nextReleases
            };
        }
}
```

```
        return result;
    }

[HttpGet("{id:int}"), AllowAnonymous]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }

    //TODO: Sistema de votación.
    var votesMedia = 4;
    var userVote = 5;

    var model = new MovieViewDTO();
    model.Movie = movie;
    model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
    model.Actors = movie.MovieActor.Select(ma => new Actor {
        Name = ma.Actor!.Name,
        Photo = ma.Actor.Photo,
        Character = ma.Character,
        ID = ma.Actor.ID
    }).ToList();

    model.VotesMedia = votesMedia;
    model.UserVote = userVote;

    return model;
}

[HttpGet("filter"), AllowAnonymous]
public async Task<ActionResult<List<Movie>>> Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);
}
```

```
if(model.Releases) {
    var today = DateTime.Today;

    queryableMovies = queryableMovies
        .Where(x => x.ReleaseDate >= today);
}

if (model.GenreID != 0)
    queryableMovies = queryableMovies
        .Where(x => x.GenresMovie
            .Select(y => y.GenreID)
            .Contains(model.GenreID));

//TODO: Implementar votación

await HttpContext.InserPaginationParametersInResponse(queryableMovies,
model.RecordCount);

//Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrEmpty(movie.Poster)) {
```

```
var poster = Convert.FromBase64String(movie.Poster);
movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
}

WriteActorsOrder(movie);

context.Add(movie);
await context.SaveChangesAsync();
return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
if (movie.MovieActor is not null) {
    for (int i = 0; i < movie.MovieActor.Count; i++) {
        movie.MovieActor[i].Orden = i + 1;
    }
}
}

[HttpPost]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrEmpty(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, "jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if (movie is null)
```

```
        return NotFound();

        context.Remove(movie);    //Marcamos para borrar el actor
        await context.SaveChangesAsync();
        if (!string.IsNullOrWhiteSpace(movie.Poster))
            await fileSaver.DeleteFile(movie.Poster!, container);

        return NoContent();      //Todo se hizo correctamente
    }
}
}
```

Con esos 4 cambios, conseguimos que las apis sigan funcionando aún sin un usuario logueado pero que no se puedan consumir las demás apis.

Hacemos lo mismo para las controllers de géneros y actores. Para el primero, necesitaremos que el Get tenga acceso anónimo porque se lo necesita para los filtros:

GenreController.cs

```
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpGet, AllowAnonymous]
        public async Task<ActionResult<IEnumerable<Genre>>> Get() {
            return await context.Genres.ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Genre>> Get(int id) {
            var genre = await context.Genres.FirstOrDefaultAsync(genre => genre.ID == id);

            if (genre is null)
                return NotFound();
        }
    }
}
```

```
        return genre;
    }

[HttpPost]
public async Task<ActionResult<int>> Post(Genre genre) {
    context.Add(genre);
    await context.SaveChangesAsync();
    return genre.ID;
}

[HttpPut]
public async Task<ActionResult<int>> Put(Genre genre) {
    context.Update(genre); //Marco el género para ser actualizado.
    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent(); //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var affectedFiles = await context.Genres
        .Where(x => x.ID == id)
        .ExecuteDeleteAsync();

    if(affectedFiles == 0)
        return NotFound();

    return NoContent();
}
}
```

ActorsController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]

    public class ActorsController: ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
```

```
private readonly IMapper mapper;
private readonly string container = "people";

public ActorsController(ApplicationDbContext context, IFileSaver fileSaver, IMapper mapper) {
    this.context = context;
    this.fileSaver = fileSaver;
    this.mapper = mapper;
}

[HttpGet]
public async Task<ActionResult<IEnumerable<Actor>>> Get([FromQuery] PaginationDTO
pagination) {
    //return await context.Actors.ToListAsync();
    var queryable = context.Actors.AsQueryable();
    await HttpContext.InserPaginationParametersInResponse(queryable, pagination.RecordCount);
    return await queryable.OrderBy(x => x.Name).ToPage(pagination).ToListAsync();
}

[HttpGet("search/{searchText}")]
public async Task<ActionResult<IEnumerable<Actor>>> Get(string searchText) {
    if(string.IsNullOrWhiteSpace(searchText))
        return new List<Actor>();

    searchText = searchText.ToLower();
    return await context.Actors
        .Where(x => x.Name.ToLower().Contains(searchText))
        .Take(5)
        .ToListAsync();
}

[HttpGet("{id:int}")]
public async Task<ActionResult<Actor>> Get(int id) {
    var actor = await context.Actors.FirstOrDefaultAsync(actor => actor.ID == id);

    if(actor is null)
        return NotFound();

    return actor;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Actor actor) {
    if(!string.IsNullOrWhiteSpace(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actor.Photo = await fileSaver.SaveFile(photoActor, "jpg", container);
    }

    context.Add(actor);
    await context.SaveChangesAsync();
}
```

```
        return actor.ID;
    }

    [HttpPost]
    public async Task<ActionResult> Put(Actor actor) {
        var actorDB = await context.Actors.FirstOrDefaultAsync(a => a.ID == actor.ID);

        if(actorDB is null)
            return NotFound();
        //Tomá las propiedades de actor y pasalas a actorDB
        actorDB = mapper.Map(actor, actorDB);

        if(!string.IsNullOrWhiteSpace(actor.Photo)) {
            //Nos mandaron una foto desde el frontend
            var photoActor = Convert.FromBase64String(actor.Photo);
            actorDB.Photo = await fileSaver.EditFile(photoActor, ".jpg", container, actorDB.Photo!);
        }

        await context.SaveChangesAsync(); //Se hace el UPDATE
        return NoContent();           //Todo se hizo correctamente
    }

    [HttpDelete("{id:int}")]
    public async Task<ActionResult<int>> Delete(int id) {
        var actor = await context.Actors.FirstOrDefaultAsync(x => x.ID == id);

        if(actor is null)
            return NotFound();

        context.Remove(actor); //Marcamos para borrar el actor
        await context.SaveChangesAsync();
        if(!string.IsNullOrWhiteSpace(actor.Photo))
            await fileSaver.DeleteFile(actor.Photo!, container);

        return NoContent();
    }
}
```

Componente de votación

Crearemos el componente **Rating.razor** en la carpeta **Shared** del proyecto **Client**. En él necesitamos crear la variable `numberStar` para poder pasársela como parámetro a las funciones `onClickHandler` y `onMouseOverHandler`.

Necesitaremos las fuentes de Font-awesome. Para eso buscamos font-awesome cdn en Google y vamos al link:

<https://cdnjs.com/libraries/font-awesome>

y copiamos el primer link. En el momento de escribir este doc:

<https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css>

Y lo agregamos en el archivo index.html:

```
index.html
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
  <title>BlazorPeliculas</title>
  <base href="/" />
  <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
  <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css" rel="stylesheet" />
</head>
<link href="css/app.css" rel="stylesheet" />
<link rel="icon" type="image/png" href="favicon.png" />
<link href="BlazorPeliculas.Client.styles.css" rel="stylesheet" />
<link href="_content/Blazored.Typeahead/blazored-typeahead.css" rel="stylesheet" />
</head>

<body>
  <div id="app">
    <svg class="loading-progress">
      <circle r="40%" cx="50%" cy="50%" />
      <circle r="40%" cx="50%" cy="50%" />
    </svg>
    <div class="loading-progress-text"></div>
  </div>

  <div id="blazor-error-ui">
    An unhandled error has occurred.
    <a href="" class="reload">Reload</a>
    <a class="dismiss">X</a>
  </div>
  <script src="_framework/blazor.webassembly.js"></script>
  <script src="js/Utilities.js"></script>
  <script src="_content/CurrieTechnologies.Razor.SweetAlert2/sweetalert2.min.js"></script>
  <script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>
</body>

</html>
```

Y agregamos el estilo **checked** en el archivo **app.css**.

```
app.css
@import url('open-iconic/font/css/open-iconic-bootstrap.min.css');

html, body {
    font-family: 'Helvetica Neue', Helvetica, Arial, sans-serif;
}

h1:focus {
    outline: none;
}

a, .btn-link {
    color: #0071c1;
}

.btn-primary {
    color: #fff;
    background-color: #1b6ec2;
    border-color: #1861ac;
}

.btn:focus, .btn:active:focus, .btn-link.nav-link:focus, .form-control:focus, .form-check-input:focus {
    box-shadow: 0 0 0 0.1rem white, 0 0 0 0.25rem #258cfb;
}

.content {
    padding-top: 1.1rem;
}

.valid.modified:not([type=checkbox]) {
    outline: 1px solid #26b050;
}

.invalid {
    outline: 1px solid red;
}

.validation-message {
    color: red;
}

#blazor-error-ui {
    background: lightyellow;
    bottom: 0;
    box-shadow: 0 -1px 2px rgba(0, 0, 0, 0.2);
    display: none;
    left: 0;
    padding: 0.6rem 1.25rem 0.7rem 1.25rem;
```

```

position: fixed;
width: 100%;
z-index: 1000;
}

#blazor-error-ui .dismiss {
    cursor: pointer;
    position: absolute;
    right: 0.75rem;
    top: 0.5rem;
}

.blazor-error-boundary {
    background:
url(data:image/svg+xml;base64,PHN2ZyB3aWR0aD0iNTYiIglaWdodD0iNDkiIHhtbG5zPSJodHRwOi8vd3d3
LnczLm9yZy8yMDAwL3N2ZylgeG1sbnM6eGxpbms9lmh0dHA6Ly93d3cudzMub3JnLzE5OTkveGxpbsiG92
ZXJmbG93PSJoaWRkZW4iPjxkZWZzPjxjbGlwUGF0aCBpZD0iY2xpcDAiPjxyZWN0IHg9jIzNSlgeT0iNTEiHdpZ
HRoPSI1NilgaGVpZ2h0PSI0OSlvPjwvY2xpcFBhdGg+PC9kZWZzPjxnIGNsaXAtcGF0aD0idXJsKCNjbGlwMCkIl
HRyYW5zZm9ybT0idHJhbnNsYXRICK0yMzUgLTUxKSI+PHBhdGggZD0iTTI2My41MDYgNTFDMjY0LjcxNyA1
MSAyNjUuODEzIDUxLjQ4MzcgMjY2LjYwNiA1Mi4yNjU4TDI2Ny4wNTlgNTluNzk4NyAyNjcuNTM5IDUzLjYyOD
DMgMjkwLjE4NSA5Mi4xODMxIDI5MC41NDUgOTluNzk1IDI5MC42NTYgOTluOTk2QzI5MC44NzcgOTMuNT
EzIDI5MSA5NC4wODE1IDI5MSA5NC42NzgyIDI5MSA5Ny4wNjUxIDI4OS4wMzggOTkgMjg2LjYxNyA5OUwy
NDAuMzgzIDk5QzIzNy45NjMgOTkgMjM2IDk3LjA2NTEgMjM2IDk0LjY3ODlgMjM2IDk0LjM3OTkgMjM2LjAz
MSA5NC4wODg2IDlzNi4wODkgOTMuODA3MkwyMzYuMzM4IDkzLjAxNjlgMjM2Ljg1OCA5Mi4xMzE0IDI1O
S40NzMgNTMuNjl5NCAYNTkuOTYxIDUyLjc5ODUgMjYwLjQwNyA1Mi4yNjU4QzI2MS4yIDUxLjQ4MzcgMjYy
LjI5NiA1MSAyNjMuNTA2IDUxWk0yNjMuNTg2IDY2LjAxODNDMjYwLjczNyA2Ni4wMTgzIDI1OS4zMTMgNjc
uMTI0NSAyNTkuMzEzIDY5LjMzNyAyNTkuMzEzIDY5LjYxMDlgMjU5LjMzMiA2OS44NjA4IDI1OS4zNzEgNzAu
MDg4N0wyNjEuNzk1IDg0LjAxNjEgMjY1LjM4IDg0LjAxNjEgMjY3LjgyMSA2OS43NDc1QzI2Ny44NiA2OS43M
zA5IDI2Ny44NzkgNjkuNTg3NyAyNjcuODc5IDY5LjMxNzkgMjY3Ljg3OSA2Ny4xMTgyIDI2Ni40NDggNjYuMD
E4MyAyNjMuNTg2IDY2LjAxODNaTTI2My41NzYgODYuMDU0N0MyNjEuMDQ5IDg2LjA1NDcgMjU5Lj4NiA4
Ny4zMDA1IDI1OS43ODYgODkuNzkyMSAyNTkuNzg2IDkyLjI4MzcgMjYxLjA0OSA5My41Mjk1IDI2My41NzYg
OTMuNTI5NSAyNjYuMTE2IDkzLjUyOTUgMjY3LjM4NyA5Mi4yODM3IDI2Ny4zODcgODkuNzkyMSAyNjcuMz
g3IDg3LjMwMDUgMjY2LjExNiA4Ni4wNTQ3IDI2My41NzYgODYuMDU0N1oiLGzpbGw9liNGRkU1MDAiIGZp
bGwtnVsZT0iZXZlrb9kZClvPjwvZz48L3N2Zz4=) no-repeat 1rem/1.8rem, #b32121;
padding: 1rem 1rem 1rem 3.7rem;
color: white;
}

.blazor-error-boundary::after {
    content: "An error has occurred."
}

.loading-progress {
    position: relative;
    display: block;
    width: 8rem;
    height: 8rem;
    margin: 20vh auto 1rem auto;
}

```

```
.loading-progress circle {  
    fill: none;  
    stroke: #e0e0e0;  
    stroke-width: 0.6rem;  
    transform-origin: 50% 50%;  
    transform: rotate(-90deg);  
}  
  
.loading-progress circle:last-child {  
    stroke: #1b6ec2;  
    stroke-dasharray: calc(3.141 * var(--blazor-load-percentage, 0%) * 0.8), 500%;  
    transition: stroke-dasharray 0.05s ease-in-out;  
}  
  
.loading-progress-text {  
    position: absolute;  
    text-align: center;  
    font-weight: bold;  
    inset: calc(20vh + 3.25rem) 0 auto 0.2rem;multiple-selector  
}  
  
.loading-progress-text:after {  
    content: var(--blazor-load-percentage-text, "Loading");  
}  
  
.form-markdown {  
    display:flex;  
}  
  
.form-markdown textarea {  
    width:500px;  
    height:500px;  
    margin-right:15px;  
}  
  
.form-markdown .markdown-container {  
    border: 1px dashed black;  
    width: 500px;  
    height: 500px;  
}  
  
.multiple-selector {  
    display:flex;  
}  
  
.selectable-ul {  
    height:200px;  
    overflow-y:auto;  
    list-style-type:none;  
    width:170px;
```

```
padding:0;
border-radius:3px;
border:1px solid #ccc;
}

.selectable-ul li {
  cursor:pointer;
  border-bottom:1px #eee solid;
  padding:2px 10px;
  font-size:14px;
}

.selectable-ul li:hover {
  background-color:#08c;
}

.multiple-selector-buttons {
  display:flex;
  flex-direction:column;
  justify-content:center;
  padding:5px;
}

.multiple-selector-buttons button {
  margin:5px;
}

.checked {
  color:orange;
}
```

Rating.razor

```
@for(int i = 1; i <= MaxPoints; i++) {
  var numberStar = i;
  <span
    @onclick="@(() => onClickHandler(numberStar))"
    @onmouseover="@(() => onMouseOverHandler(numberStar))"
    style="cursor:pointer"
    class="fa fa-star @(SelectedPoint >= i ? " checked" : null)"
  ></span>
}

@code {
  [Parameter] public int MaxPoints { get; set; }
  [Parameter] public int SelectedPoint { get; set; }
  [Parameter] public EventCallback<int> OnRating { get; set; }
  private bool voted = false;

  private async Task onClickHandler(int numberStar) {
```

```

SelectedPoint = numberStar;
voted = true;
await OnRating.InvokeAsync(SelectedPoint);
}

private async Task onMouseOverHandler(int numberStar) {
    if(!voted)
        SelectedPoint = numberStar;
}
}
    
```

ViewMovie.razor

```

@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
@inject IRepository repository
@inject SweetAlertService swal

@if(model is null) {
    <LoadingWheel />
}
else {
    <h2>@movie.Title (@movie.ReleaseDate!.Value.ToString("yyyy"))</h2>
    @foreach(var genre in model.Genres) {
        <a class="me-2 badge bg-primary rounded-pill text-decoration-none"
        href="movies/filter?genreid=@genre.ID">@genre.Name</a>
    }

    <span>| @movie.ReleaseDate!.Value.ToString("dd MM yyyy")
    | Media: @model.VotesMedia.ToString("0.#")/5
    | Your vote: <Rating MaxPoints="5" SelectedPoint="model.UserVote"></Rating></span>

    <div class="d-flex mt-2">
        <span style="display:inline-block;" class="me-2">
            
        </span>

        <iframe width="560" height="315" src="https://www.youtube.com/embed/@movie.Trailer"
        title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write;
        encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
    </div>
    <div class="mt-2">
        <h3>Summary</h3>
        <div>
            <ShowMD MDContent="@movie.Summary" />
        </div>
    </div>

    <div class="mt-2">
        <h3>Actors</h3>
    </div>
}
    
```

```
<div class="d-flex flex-column">
    @foreach(var actor in model.Actors) {
        <div class="mb-2">
            
            <span style="display:inline-block;width:200px;">@actor.Name</span>
            <span style="display:inline-block;width:45px;">...</span>
            <span>@actor.Character</span>
        </div>
    }
</div>
</div>
}

@code {
    [Parameter] public int MovieID { get; set; }
    [Parameter] public string MovieName { get; set; } = null!;
    private MovieViewDTO? model;
    private Movie movie = null!;

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<MovieViewDTO>($"api/movies/{MovieID}");

        if (responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            model = responseHTTP.Response;
            movie = model!.Movie;
        }
    }
}
```

Vamos a visualizar la película y vemos al componente de Rating en acción. Al principio aparecen todas prendidas porque en MoviesController está hardcodeado que la votación del usuario es 5. Sin embargo, si movemos el mouse por sobre las estrellas se van pintando/despintando como corresponde:

Hola, emiliano@blazor.com! [Log out](#) [A](#)

Spiderman: Far from home (2019)

Action Adventure Sci-Fi | 26 06 2019 | Media: 4/5 | Your vote: ★★★★☆

Summary

Summary

In Ixtenco, Mexico, Nick Fury and Maria Hill investigate an unnatural storm and encounter the Earth Element. Quentin Beck, a super-powered individual, arrives to defeat the creature, and is subsequently recruited by Fury and Hill. In New York City, the Midtown School of Science and Technology completes its year, which was

Back-end del componente

Lo primero que necesitamos es que la entidad de votación debe tener el ID del usuario:

VoteMovie.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class VoteMovie {
        public int ID { get; set; }
        public int Voto { get; set; }
        public DateTime VoteWhen { get; set; }
        public int MovieID { get; set; }
        public Movie? Movie { get; set; }
        public string UserID { get; set; } = null!;
    }
}
```

Luego de eso, abrimos el ApplicationDbContext.cs para agregar esta entidad y poder pedirle a Entity Framework Core que nos cree la nueva tabla:

ApplicationContext.cs

```

using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : IdentityDbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder); //No se puede eliminar esta línea.

            modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });
            modelBuilder.Entity<MovieActor>().HasKey(x => new { x.ActorID, x.MovieID });
        }

        public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase Genre.
        public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
        public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase Movie.
        public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla GenresMovie a partir de la clase GenresMovie.
        public DbSet<MovieActor> MoviesActors => Set<MovieActor>(); //Creamos la tabla MoviesActors a partir de la clase MovieActor.
        public DbSet<VoteMovie> VotesMovies => Set<VoteMovie>(); //Creamos la tabla VotesMovies a partir de la clase VoteMovie.
    }
}

```

Creamos la migración con los siguientes comandos:

PMC	EFC cli
Add-Migration VoteMovie	dotnet ef migrations add VoteMovie
Update-database	dotnet ef database update

```

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any
licenses to, third-party packages. Some packages may include dependencies which are governed by
additional licenses. Follow the package source (feed) URL to determine any dependencies.

```

```

Package Manager Console Host Version 6.4.0.111

```

```

Type 'get-help NuGet' to see all available NuGet commands.

```

```

PM> Add-Migration VotesMovies
Build started...
Build succeeded.

```

```
To undo this action, use Remove-Migration.
PM> update-database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (14ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'__EFMigrationsHistory');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'__EFMigrationsHistory');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT [MigrationId], [ProductVersion]
        FROM [__EFMigrationsHistory]
        ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20230216185633_VotesMovies'.
Applying migration '20230216185633_VotesMovies'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE TABLE [VotesMovies] (
        [ID] int NOT NULL IDENTITY,
        [Voto] int NOT NULL,
        [VoteWhen] datetime2 NOT NULL,
        [MovieID] int NOT NULL,
        [UserID] nvarchar(max) NOT NULL,
        CONSTRAINT [PK_VotesMovies] PRIMARY KEY ([ID]),
        CONSTRAINT [FK_VotesMovies_MovieID] FOREIGN KEY ([MovieID]) REFERENCES [Movies]
        ([ID]) ON DELETE CASCADE
    );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE INDEX [IX_VotesMovies_MovieID] ON [VotesMovies] ([MovieID]);
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
    VALUES (N'20230216185633_VotesMovies', N'7.0.2');
Done.
PM>
PM>
```

El primer comando crea el archivo 20230216185633_VotesMovies.cs que nos muestra que sólo crearía la tabla VotesMovies.

```
20230216185633_VotesMovies.cs
using System;
using Microsoft.EntityFrameworkCore.Migrations;

#nullable disable

namespace BlazorPeliculas.Server.Migrations
{
    /// <inheritdoc />
    public partial class VotesMovies : Migration
    {
```

```
/// <inheritdoc />
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "VotesMovies",
        columns: table => new
        {
            ID = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Voto = table.Column<int>(type: "int", nullable: false),
            VoteWhen = table.Column<DateTime>(type: "datetime2", nullable: false),
            MovieID = table.Column<int>(type: "int", nullable: false),
            UserID = table.Column<string>(type: "nvarchar(max)", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_VotesMovies", x => x.ID);
            table.ForeignKey(
                name: "FK_VotesMovies_Movies_MovieID",
                column: x => x.MovieID,
                principalTable: "Movies",
                principalColumn: "ID",
                onDelete: ReferentialAction.Cascade);
        });
}

migrationBuilder.CreateIndex(
    name: "IX_VotesMovies_MovieID",
    table: "VotesMovies",
    column: "MovieID");
}

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "VotesMovies");
}
```

El segundo comando, ejecuta los cambios en la BD.

Ahora, crearemos la clase **VotesController.razor** para poder guardar los votos del usuario y/o devolver los puntajes de la película solicitada. Pero antes, tendremos que crear un DTO llamado **VoteMovieDTO.cs** en la carpeta **DTOs** del proyecto **Shared**.

VoteMovieDTO.cs

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class VoteMovieDTO {
        public int MovieID { get; set; }
        public int Voto { get; set; }
    }
}
```

También tendremos que crear el perfil de AutoMapper para poder mapear los campos del DTO hacia la tabla.

AutoMapperProfiles.cs

```
using AutoMapper;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;

namespace BlazorPeliculas.Server.Helpers {
    public class AutoMapperProfile : Profile {
        public AutoMapperProfile() {
            CreateMap<Actor, Actor>()
                .ForMember(x => x.Photo, option => option.Ignore());

            CreateMap<Movie, Movie>()
                .ForMember(x => x.Poster, option => option.Ignore());

            CreateMap<VoteMovieDTO, VoteMovie>();
        }
    }
}
```

El código del controller, entonces, es:

VotesController.cs

```
using AutoMapper;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/votes"), Authorize(AuthenticationSchemes =
```

```
JwtBearerDefaults.AuthenticationScheme)]  
  
public class VotesController: ControllerBase {  
    private readonly ApplicationDbContext context;  
    private readonly UserManager<IdentityUser> userManager;  
    private readonly IMapper mapper;  
  
    public VotesController(ApplicationDbContext context,  
        UserManager<IdentityUser> userManager,  
        IMapper mapper) {  
        this.context = context;  
        this.userManager = userManager;  
        this.mapper = mapper;  
    }  
  
    [HttpPost]  
    public async Task<ActionResult> Vote(VoteMovieDTO voteMovieDTO) {  
        //Tomo el nombre (email) del usuario y con eso busco y obtengo el registro de dicho usuario.  
        var user = await userManager.FindByEmailAsync(HttpContext.User.Identity!.Name!);  
  
        if(user is null) {  
            //No debería pasar nunca, pero por las dudas...  
            return BadRequest("User was not found");  
        }  
  
        var userID = user.Id;  
  
        var currentVote = await context.VotesMovies  
            .FirstOrDefaultAsync(x => x.MovieID == voteMovieDTO.MovieID && x.UserID == userID);  
  
        if(currentVote is null) {  
            //Aún no voto. Crear uno nuevo.  
            var voteMovie = mapper.Map<VoteMovie>(voteMovieDTO);  
            voteMovie.UserID = userID;  
            voteMovie.VoteWhen = DateTime.Now;  
            context.Add(voteMovie); //Marcar para grabar  
        }  
        else {  
            //Ya votó. Actualizar la fecha y el voto  
            currentVote.VoteWhen = DateTime.Now;  
            currentVote.Voto = voteMovieDTO.Voto;  
        }  
  
        await context.SaveChangesAsync();  
        return NoContent();  
    }  
}
```

En el componente ViewMovie.razor tenemos que crear el callback para poder llamar al POST que grabe el valor seleccionado:

ViewMovie.razor

```

@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
@inject IRepository repository
@inject SweetAlertService swal

@if(model is null) {
    <LoadingWheel />
}
else {
    <h2>@movie.Title (@movie.ReleaseDate!.Value.ToString("yyyy"))</h2>
    @foreach(var genre in model.Genres) {
        <a class="me-2 badge bg-primary rounded-pill text-decoration-none"
        href="movies/filter?genreid=@genre.ID">@genre.Name</a>
    }

    <span>| @movie.ReleaseDate!.Value.ToString("dd MM yyyy")
    | Media: @model.VotesMedia.ToString("0.#")/5
    | Your vote: <Rating MaxPoints="5" SelectedPoint="model.UserVote"
    OnRating="OnRating"></Rating></span>

    <div class="d-flex mt-2">
        <span style="display:inline-block;" class="me-2">
            
        </span>

        <iframe width="560" height="315" src="https://www.youtube.com/embed/@movie.Trailer"
        title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write;
        encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
    </div>
    <div class="mt-2">
        <h3>Summary</h3>
        <div>
            <ShowMD MDContent="@movie.Summary" />
        </div>
    </div>

    <div class="mt-2">
        <h3>Actors</h3>
        <div class="d-flex flex-column">
            @foreach(var actor in model.Actors) {
                <div class="mb-2">
                    
                    <span style="display:inline-block;width:200px;">@actor.Name</span>
                    <span style="display:inline-block;width:45px;">...</span>
                    <span>@actor.Character</span>
                </div>
            }
        </div>
    </div>
}

```

```
        }
    </div>
</div>
}

@code {
[Parameter] public int MovieID { get; set; }
[Parameter] public string MovieName { get; set; } = null!;
private MovieViewDTO? model;
private Movie movie = null!;

protected override async Task OnInitializedAsync() {
    var responseHTTP = await repository.Get<MovieViewDTO>($"api/movies/{MovieID}");

    if (responseHTTP.Error) {
        var ErrMessage = await responseHTTP.GetErrMsg();
        await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
    }
    else {
        model = responseHTTP.Response;
        movie = model!.Movie;
    }
}

private async Task OnRating(int selectedVote) {
    model!.UserVote = selectedVote;
    var voteMovieDTO = new VoteMovieDTO() {
        MovieID = MovieID,
        Vote = selectedVote
    };

    var responseHTTP = await repository.Post("api/votes", voteMovieDTO);

    if(responseHTTP.Error) {
        var ErrMessage = await responseHTTP.GetErrMsg();
        await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
    }
    else
        await swAl.FireAsync("Success!", "Your vote has been received.", SweetAlertIcon.Success);
}
}
```

Finalmente, necesitamos modificar el MoviesController para obtener la información real de la votación de la película:

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
```

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly UserManager<IdentityUser> userManager;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper,
            UserManager<IdentityUser> userManager) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
            this.userManager = userManager;
        }

        [HttpGet, AllowAnonymous]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
                .Take(limit)
                .OrderBy(movie => movie.ReleaseDate)
                .ToListAsync();

            var result = new HomePageDTO {
                OnBoard = onBoardMovies,
                NextReleases = nextReleases
            };
            return result;
        }
    }
}
```

```
[HttpGet("{id:int}"), AllowAnonymous]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }

    var votesMedia = 0.0;
    var userVote = 0;

    if(await context.VotesMovies.AnyAsync(x => x.MovieID == id)) {
        //Alguien ha votado por la película
        votesMedia = await context.VotesMovies
            .Where(x => x.MovieID == id)
            .AverageAsync(x => x.Voto);

        if(HttpContext.User.Identity!.IsAuthenticated) {
            var user = await userManager.FindByEmailAsync(HttpContext.User.Identity!.Name!);

            if(user is null) {
                //No debería pasar nunca, pero por las dudas...
                return BadRequest("User was not found");
            }

            var userID = user.Id;

            var userVoteDB = await context.VotesMovies
                .FirstOrDefaultAsync(x => x.MovieID == id && x.UserID == userID);

            if(userVoteDB is not null)
                userVote = userVoteDB.Voto;
        }
    }

    var model = new MovieViewDTO();
    model.Movie = movie;
    model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
    model.Actors = movie.MovieActor.Select(ma => new Actor {
        Name = ma.Actor!.Name,
        Photo = ma.Actor.Photo,
        Character = ma.Character,
        ID = ma.Actor.ID
    });
}
```

```
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("filter"), AllowAnonymous]
public async Task<ActionResult<List<Movie>>> Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);

    if(model.Releases) {
        var today = DateTime.Today;

        queryableMovies = queryableMovies
            .Where(x => x.ReleaseDate >= today);
    }

    if (model.GenreID != 0)
        queryableMovies = queryableMovies
            .Where(x => x.GenresMovie
                .Select(y => y.GenreID)
                .Contains(model.GenreID));

    //TODO: Implementar votación

    await HttpContext.InserPaginationParametersInResponse(queryableMovies,
model.RecordCount);

    //Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
    var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
    return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
```

```
return NotFound();

var movieViewDTO = movieActionResult.Value;      //Será el DTO
var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
var unselectedGenres = await context.Genres
    .Where(x => !selectedGenresIDs.Contains(x.ID))
    .ToListAsync();

var model = new MovieUpdateDTO();
model.Movie = movieViewDTO.Movie;
model.UnselectedGenres = unselectedGenres;
model.SelectedGenres = movieViewDTO.Genres;
model.Actors = movieViewDTO.Actors;

return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPut]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
}
```

```

movieDB = mapper.Map(movie, movieDB);

if (!string.IsNullOrEmpty(movie.Poster)) {
    //Nos mandaron una foto desde el frontend
    var Poster = Convert.FromBase64String(movie.Poster);
    movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
}

WriteActorsOrder(movieDB);

await context.SaveChangesAsync(); //Se hace el UPDATE
return NoContent();           //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if (movie is null)
        return NotFound();

    context.Remove(movie); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if (!string.IsNullOrEmpty(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

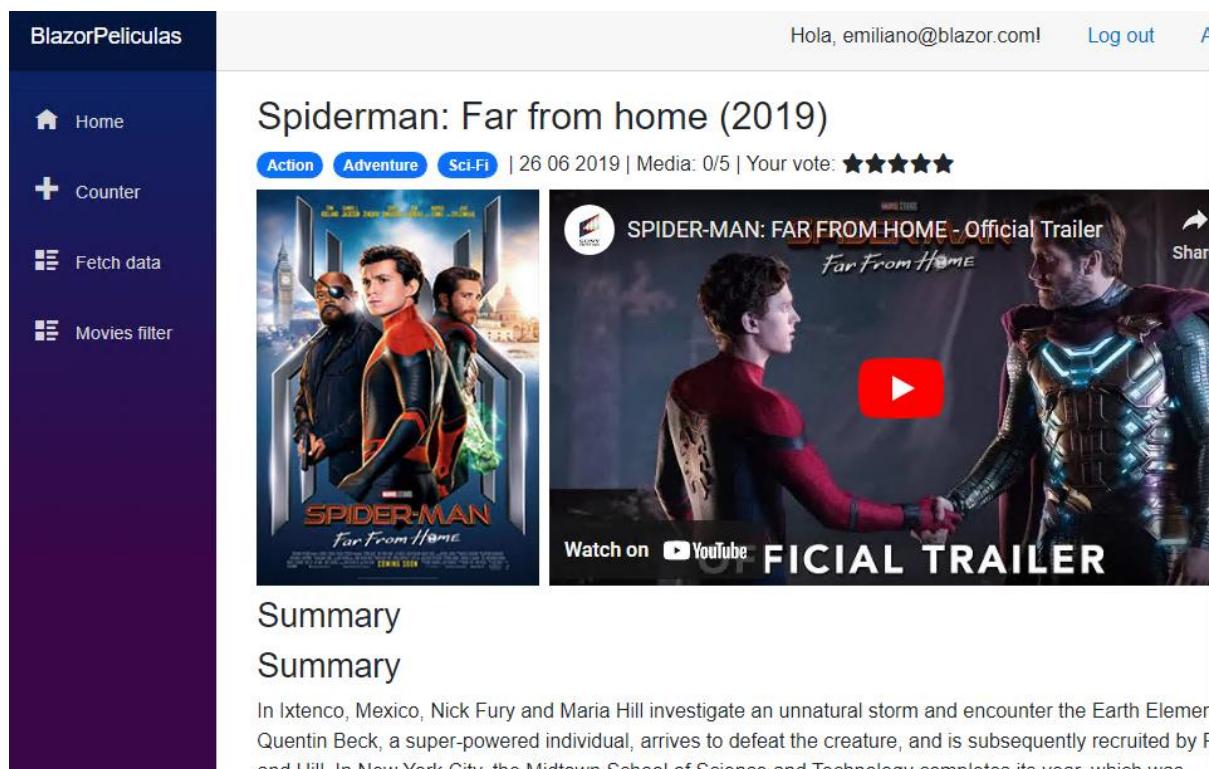
    return NoContent();   //Todo se hizo correctamente
}
}
}
}

```

Antes de ejecutar, vemos que la tabla está vacía:

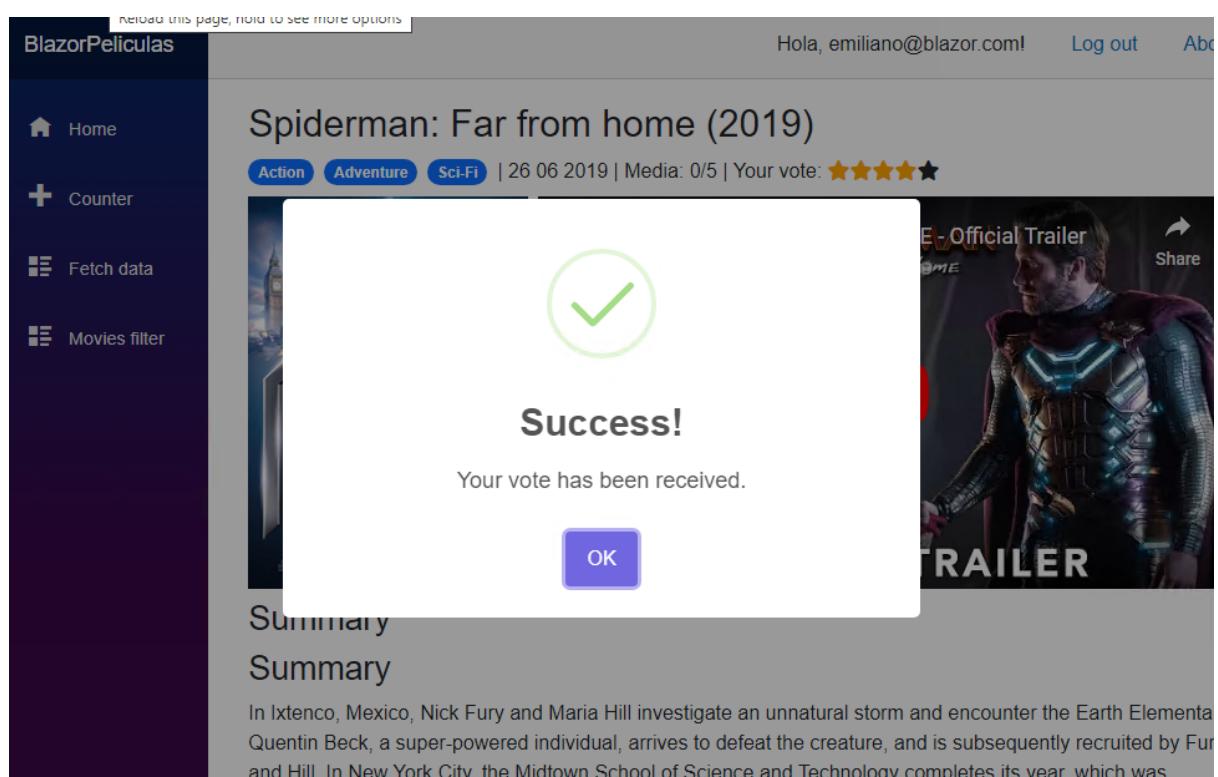
dbo.VotesMovies [Data] ➔ X MoviesController.cs					
	ID	Voto	VoteWhen	MovieID	UserID
*	NULL	NULL	NULL	NULL	NULL

Si vamos a la película, vemos que no tiene votos (0/5) y tampoco la hemos votado (0 estrellas):



The screenshot shows a Blazor application interface. On the left is a dark sidebar with the title "BlazorPelículas" and four menu items: "Home", "Counter", "Fetch data", and "Movies filter". The main content area displays information for the movie "Spiderman: Far from home (2019)". At the top, there are genre tags: Action, Adventure, Sci-Fi. Below them is the release date: 26 06 2019. The media rating is listed as 0/5, and the user's current vote is indicated by five empty star icons. To the right of this text is a movie poster featuring Spider-Man and other characters. Below the poster is a thumbnail for the "Official Trailer" on YouTube, which includes a play button icon. The trailer thumbnail features a scene from the movie where Spider-Man and Doctor Strange are interacting. At the bottom of the main content area, there are two "Summary" sections followed by a truncated movie plot.

Le damos una votación de 4 y recibimos el mensaje de éxito:



Y en la tabla se guardó el voto:

ViewMovie.razor						dbo.VotesMovies [Data]	X
	Max Rows:	1000	▼	▼	▼	▼	▼
ID	Voto	VoteWhen	MovielD	UserID			
1	4	16/02/2023 16:48:20	1	3e96335b-2a79-412d-8eb3-d05f5f424259	NULL	NULL	NULL

Al refrescar, vemos que la película ya tiene votación (4/5) y nos muestra la puntuación que le dimos (4 estrellas).

The screenshot shows a Blazor application interface. On the left is a sidebar with a dark purple background and white icons for navigation: Home, Counter, Fetch data, and Movies filter. The main content area has a light blue header bar with the text "Hola, emiliano@blazor.com!" and "Log out". Below the header, the title "Spiderman: Far from home (2019)" is displayed, along with genre filters (Action, Adventure, Sci-Fi) and release date (26 06 2019). A media rating of 4/5 is shown with a yellow star icon. The main content includes the movie poster for "SPIDER-MAN: FAR FROM HOME - Official Trailer", a YouTube video player for the trailer, and a summary section with the heading "Summary". Below the summary, there is a truncated text block.

Filtro de más votadas

Lo primero que haremos es agregar una propiedad de navegación en la entidad de **Movie.cs**.

Movie.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Movie {
        public int ID { get; set; }
        [Required]
        public string Title { get; set; } = null!;
        public string? Summary { get; set; }
        public bool OnBillboard { get; set; }
        public string? Trailer { get; set; }
        public DateTime? ReleaseDate { get; set; }
        public string? Poster { get; set; }
        public List<GenresMovie> GenresMovie { get; set; } = new List<GenresMovie>();
        public List<MovieActor> MovieActor { get; set; } = new List<MovieActor>();
        public List<VoteMovie> VotesMovies { get; set; } = new List<VoteMovie>();
        public string? TrimmedTitle {
            get {
                return Title?.Trim();
            }
        }
    }
}
```

```
if(string.IsNullOrWhiteSpace(title)) {
    return null;
}

if(title.Length > 60) {
    return title.Substring(0, 60) + "...";
}
else {
    return title;
}
}

public string? urlTitle() {
    return title.Replace(" ", "-");
}
}
```

En el controlador de películas seguiremos con nuestro modelo de ejecución diferida. Tomaremos el **queryableMovies** y agregaremos el **OrderByDescending** para indicar que debe ordenar según el promedio de votación de cada película.

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly UserManager<IdentityUser> userManager;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper,
            UserManager<IdentityUser> userManager) {
```

```
this.context = context;
this.fileSaver = fileSaver;
this.mapper = mapper;
this.userManager = userManager;
}

[HttpGet, AllowAnonymous]
public async Task<ActionResult<HomePageDTO>> Get() {
    var limit = 6;
    var onBoardMovies = await context.Movies
        .Where(movie => movie.OnBillboard)
        .Take(limit)
        .OrderByDescending(movie => movie.ReleaseDate)
        .ToListAsync();
    var today = DateTime.Today;
    var nextReleases = await context.Movies
        .Where(movie => movie.ReleaseDate > today)
        .Take(limit)
        .OrderBy(movie => movie.ReleaseDate)
        .ToListAsync();

    var result = new HomePageDTO {
        OnBoard = onBoardMovies,
        NextReleases = nextReleases
    };
    return result;
}

[HttpGet("{id:int}"), AllowAnonymous]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }

    var votesMedia = 0.0;
    var userVote = 0;

    if(await context.VotesMovies.AnyAsync(x => x.MovieID == id)) {
        //Alguien ha votado por la película
        votesMedia = await context.VotesMovies
            .Where(x => x.MovieID == id)
```

```
.AverageAsync(x => x.Voto);

if(HttpContext.User.Identity!.IsAuthenticated) {
    var user = await userManager.FindByEmailAsync(HttpContext.User.Identity!.Name!);

    if(user is null) {
        //No debería pasar nunca, pero por las dudas...
        return BadRequest("User was not found");
    }

    var userID = user.Id;

    var userVoteDB = await context.VotesMovies
        .FirstOrDefaultAsync(x => x.MovieID == id && x.UserID == userID);

    if(userVoteDB is not null)
        userVote = userVoteDB.Voto;
}

var model = new MovieViewDTO();
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("filter"), AllowAnonymous]
public async Task<ActionResult<List<Movie>> > Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);

    if(model.Releases) {
```

```

var today = DateTime.Today;

queryableMovies = queryableMovies
    .Where(x => x.ReleaseDate >= today);
}

if (model.GenreID != 0)
    queryableMovies = queryableMovies
        .Where(x => x.GenresMovie
            .Select(y => y.GenreID)
            .Contains(model.GenreID));

if(model.MostVoted) {
    queryableMovies = queryableMovies.OrderByDescending(m => m.VotesMovies.Average(vm
=> vm.Voto));
}

await HttpContext.InserPaginationParametersInResponse(queryableMovies,
model.RecordCount);

//Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
}

```

```
if (!string.IsNullOrWhiteSpace(movie.Poster)) {
    var poster = Convert.FromBase64String(movie.Poster);
    movie.Poster = await fileSaver.SaveFile(poster, ".jpg", container);
}

WriteActorsOrder(movie);

context.Add(movie);
await context.SaveChangesAsync();
return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPost]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);
```

```

if (movie == null)
    return NotFound();

context.Remove(movie);      //Marcamos para borrar el actor
await context.SaveChangesAsync();
if (!string.IsNullOrWhiteSpace(movie.Poster))
    await fileSaver.DeleteFile(movie.Poster!, container);

return NoContent();          //Todo se hizo correctamente
}
}
}
    
```

Vemos que Spiderman tiene una puntuación de 4 mientras que Inception, de 5.

dbo.VotesMovies [Data] ➔ X MoviesController.cs					
	ID	Voto	VoteWhen	MovieID	UserID
▶	1	4	16/02/2023 17:0...	1	3e96335b-2a79...
	2	5	16/02/2023 17:1...	2	3e96335b-2a79...
*	NULL	NULL	NULL	NULL	NULL

En el listado original, primero aparece Spiderman y luego Inception:

The screenshot shows a Blazor application interface. On the left is a sidebar with navigation links: Home, Counter, Fetch data, and Movies filter (which is currently selected). The main content area is titled "Movies filter". It contains a search form with fields for "Movie title" and "Genre" (with a dropdown menu showing "Select a gen"). There are also three checkboxes: "Future premieres", "On billboard", and "Most voted". Below the form are two movie posters: "Spider-Man: Far from home" and "Inception". At the bottom of the main area, there are navigation buttons for "Previous", "1", and "Next".

Pero, si pedimos por las más votadas, vemos que primero muestra Inception (5 estrellas) y luego Spiderman (4 estrellas).

The screenshot shows a Blazor application interface. On the left is a dark sidebar with navigation links: Home, Counter, Fetch data, and Movies filter (which is highlighted). The main content area has a header with 'Hola, emiliano@blazor.com!' and links for Log out and About. Below the header is a 'Movies filter' section with a search bar for 'Movie title', a dropdown for 'Select a gen', and three checkboxes: 'Future premieres' (unchecked), 'On billboard' (unchecked), and 'Most voted' (checked). There are 'Filter' and 'Clean' buttons. Below the filter section are buttons for 'Previous', '1', and 'Next'. Two movie posters are displayed: 'Inception' and 'Spider-Man: Far from home'.

Listado de usuarios

Primero crearemos un DTO para los usuarios. Lo hacemos creando la clase **UserDTO.cs** en la carpeta **DTOs** del proyecto **Shared**.

```
UserDTO.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class UserDTO {
        public string ID { get; set; } = null!;
        public string Email { get; set; } = null!;

    }
}
```

Creamos la clase **UsersController.cs** en la carpeta **Controllers** del proyecto **Server**. La hacemos heredar de **ControllerBase**. Haremos la api de **GET** para poder traer el listado de usuarios de manera paginada.

UsersController.cs

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/users"), Authorize(AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme)]
    public class UsersController : ControllerBase {
        private readonly ApplicationDbContext context;

        public UsersController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpGet]
        public async Task<ActionResult<List<UserDTO>>> Get([FromQuery] PaginationDTO pagination) {
            var queryable = context.Users.AsQueryable();
            await HttpContext.InserPaginationParametersInResponse(queryable,
                pagination.RecordCount);
            return await queryable.ToPage(pagination)
                .Select(x => new UserDTO { ID = x.Id, Email = x.Email! })
                .ToListAsync();
        }
    }
}
```

Creamos el componente **UsersList.razor** en la carpeta **Pages/Users** del proyecto **Client**.

UsersList.razor

```
@page "/users"
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swal

<h3>Users List</h3>

<Pagination ActualPage="currentPage" TotalPages="totalPages"
SelectedPage="selectedPage"></Pagination>

<GenericList List="Users">
```

```
<HasRecordsComplete>
    <table class="table">
        <thead>
            <tr>
                <th></th>
                <th>User</th>
            </tr>
        </thead>
        <tbody>
            @foreach(var user in Users!) {
                <tr>
                    <td>
                        <a href="/users/edit/@user.ID" class="btn btn-success">Edit</a>
                    </td>
                    <td>
                        @user.Email
                    </td>
                </tr>
            }
        </tbody>
    </table>
</HasRecordsComplete>
</GenericList>

@code {
    List<UserDTO>? Users;
    private int currentPage = 1;
    private int totalPages;

    protected override async Task OnInitializedAsync() {
        await selectedPage(1);
    }

    private async Task selectedPage(int page) {
        currentPage = page;
        await Load(page);
    }

    private async Task Load(int page = 1) {
        var responseHTTP = await repository.Get<List<UserDTO>>($"api/users?page={page}");

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            totalPages = int.Parse(responseHTTP HttpResponseMessage.Headers
                .GetValues("totalPages").FirstOrDefault()!);
            Users = responseHTTP.Response;
        }
    }
}
```

```
}
```

Agregamos un punto de entrada al menú lateral para poder acceder al listado de usuarios:

NavBar.razor

```
<div class="top-row ps-3 navbar navbar-dark">
<div class="container-fluid">
    <a class="navbar-brand" href="">BlazorPelículas</a>
    <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
        <span class="navbar-toggler-icon"></span>
    </button>
</div>
</div>

<div class="@NavControllerCssClass nav-scrollable" @onclick="ToggleNavMenu">
<nav class="flex-column">
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
            <span class="oi oi-home" aria-hidden="true"></span> Home
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="counter">
            <span class="oi oi-plus" aria-hidden="true"></span> Counter
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="fetchdata">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/filter">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Movies filter
        </NavLink>
    </div>

    <div class="nav-item px-3">
        <AuthorizeView Roles="admin">
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="genres">
                    <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
                </NavLink>
            </div>
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="actors">
                    <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
                </NavLink>
            </div>
        </AuthorizeView>
    </div>
</nav>
</div>
```

```

        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/create">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
        </NavLink>
    </div>
</AuthorizeView>

<div class="nav-item px-3">
    <NavLink class="nav-link" href="users">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Users
    </NavLink>
</div>
</nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}

```

The screenshot shows a Blazor application interface. On the left, there is a dark sidebar menu with the following items:

- Home
- Counter
- Fetch data
- Movies filter
- Users

The "Users" item is highlighted with a purple background. The main content area has a light gray background and displays the following:

Users List

Navigation buttons: Previous, 1, Next

User
emiliano@blazor.com

Buttons: Edit

Roles

Crearemos los roles mediante una migración. Así, siempre tendremos un rol creado.

PMC	EFC cli
Add-Migration RoleAdmin	dotnet ef migrations add RoleAdmin

```
PM> Add-Migration RoleAdmin
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM>
```

Esto creará una migración vacía:

```
20230216211409_RoleAdmin.cs
using Microsoft.EntityFrameworkCore.Migrations;

(nullable disable

namespace BlazorPeliculas.Server.Migrations
{
    /// <inheritdoc />
    public partial class RoleAdmin : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
        {

        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
        {
        }
    }
}
```

La idea, ahora, es armar la migración manualmente. Para generar el ID lo haremos en la página <https://guidgenerator.com/>. Tenemos que asegurarnos de que hyphens esté seleccionado.

```
20230216211409_RoleAdmin.cs
using Microsoft.EntityFrameworkCore.Migrations;

(nullable disable

namespace BlazorPeliculas.Server.Migrations
{
    /// <inheritdoc />
    public partial class RoleAdmin : Migration
    {
        /// <inheritdoc />
        protected override void Up(MigrationBuilder migrationBuilder)
```

```
{
    migrationBuilder.Sql(@"INSERT INTO AspNetRoles (Id, Name, NormalizedName)
        VALUES('505630f9-a640-44c9-840a-ed733cdc165b', 'admin', 'ADMIN');");
}

/// <inheritdoc />
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.Sql("DELETE AspNetRoles WHERE Id = '505630f9-a640-44c9-840a-
ed733cdc165b');");
}
}
```

Verificamos que la tabla está vacía:

dbo.AspNetRoles [Data]					20230216211409_RoleAdmin.cs	UsersList.razor
	Id	Name	NormalizedNames	ConcurrencyStamp		
*	NULL	NULL	NULL	NULL		

Y ahora, actualizaremos la BD:

PMC	EFC cli
Update-database	dotnet ef database update

```

PM> update-database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (13ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT OBJECT_ID(N'[_EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT OBJECT_ID(N'[_EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (4ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT [MigrationId], [ProductVersion]
    FROM [_EFMigrationsHistory]
    ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20230216211409_RoleAdmin'.
    Applying migration '20230216211409_RoleAdmin'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    INSERT INTO AspNetRoles (Id, Name, NormalizedName)
```

```

        VALUES('505630f9-a640-44c9-840a-ed733cdc165b',      'admin',
'ADMIN');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
    VALUES (N'20230216211409_RoleAdmin', N'7.0.2');

Done.
PM>

```

Esto hará el INSERT indicado:

dbo.AspNetRoles [Data] 20230216211409_RoleAdmin.cs UsersList.razor				
	Id	Name	NormalizedName	ConcurrencyStamp
▶	50a-ed733cdc165b	admin	ADMIN	NULL
*	NULL	NULL	NULL	NULL

Crearemos el DTO llamado **RoleDTO.cs** en la carpeta **DTOs** del proyecto **Shared**.

RoleDTO.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class RoleDTO {
        public string Name { get; set; } = null!;
    }
}

```

Actualizamos el UsersController.cs para obtener el listado de roles.

UsersController.cs

```

using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/users"), Authorize(AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme)]
    public class UsersController : ControllerBase {
        private readonly ApplicationDbContext context;

```

```
public UsersController(ApplicationDbContext context) {
    this.context = context;
}

[HttpGet]
public async Task<ActionResult<List<UserDTO>>> Get([FromQuery] PaginationDTO pagination) {
    var queryable = context.Users.AsQueryable();
    await HttpContext.InserPaginationParametersInResponse(queryable,
        pagination.RecordCount);
    return await queryable.ToPage(pagination)
        .Select(x => new UserDTO { ID = x.Id, Email = x.Email! })
        .ToListAsync();
}

[HttpGet("roles")]
public async Task<ActionResult<List<RoleDTO>>> Get() {
    return await context.Roles.Select(x => new RoleDTO { Name = x.Name }).ToListAsync();
}
}
```

Crearemos el DTO llamado **EditRolDTO.cs** en la carpeta **DTOs** del proyecto **Shared**.

EditRolDTO.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.DTOs {
    public class EditRolDTO {
        public string UserID { get; set; } = null!;
        public string Role { get; set; } = null!;
    }
}
```

Agregamos los métodos Post para asignar y remover un rol.

UsersController.cs

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
```

```
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/users"), Authorize(AuthenticationSchemes =
    JwtBearerDefaults.AuthenticationScheme)]
    public class UsersController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly UserManager<IdentityUser> userManager;

        public UsersController(ApplicationDbContext context,
            UserManager<IdentityUser> userManager) {
            this.context = context;
            this.userManager = userManager;
        }

        [HttpGet]
        public async Task<ActionResult<List<UserDTO>>> Get([FromQuery] PaginationDTO pagination) {
            var queryable = context.Users.AsQueryable();
            await HttpContext.InserPaginationParametersInResponse(queryable,
                pagination.RecordCount);
            return await queryable.ToPage(pagination)
                .Select(x => new UserDTO { ID = x.Id, Email = x.Email! })
                .ToListAsync();
        }

        [HttpGet("roles")]
        public async Task<ActionResult<List<RoleDTO>>> Get() {
            return await context.Roles.Select(x => new RoleDTO { Name = x.Name }).ToListAsync();
        }

        [HttpPost("assignRole")]
        public async Task<ActionResult> AssignRoleToUser(EditRoleDTO editRoleDTO) {
            var user = await userManager.FindByIdAsync(editRoleDTO.UserID);
            if(user is null)
                return BadRequest("User was not found!");

            await userManager.AddToRoleAsync(user, editRoleDTO.Role);
            return NoContent();
        }

        [HttpPost("removeRole")]
        public async Task<ActionResult> RemoveRoleFromUser(EditRoleDTO editRoleDTO) {
            var user = await userManager.FindByIdAsync(editRoleDTO.UserID);
            if(user is null)
                return BadRequest("User was not found!");

            await userManager.RemoveFromRoleAsync(user, editRoleDTO.Role);
            return NoContent();
        }
    }
}
```

{}

Creamos el componente **EditUser.razor** en la carpeta **Users** del proyecto **Client**.

EditUser.razor

```
@page "/users/edit/{UserID}"
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swal

<h3>Edit User</h3>

@if(roles is null) {
    <LoadingWheel/>
}
else {
    <div class="form-inline">
        <select class="form-select mb-2" @bind="selectedRole">
            <option value="0">-- Select a role--</option>
            @foreach(var rol in roles) {
                <option value="@rol.Name">@rol.Name</option>
            }
        </select>

        <button class="btn btn-info mb-2" @onclick="AssignRole">Assign role</button>
        <button class="btn btn-danger mb-2" @onclick="RemoveRole">Remove role</button>
    </div>
}

@code {
    [Parameter]
    public string UserID { get; set; } = null!;
    private List<RoleDTO>? roles;
    private string selectedRole = "0";

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<List<RoleDTO>>("api/users/roles");

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMessage();
            await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            roles = responseHTTP.Response;
        }
    }

    private async Task EditRole(string url) {
        if(selectedRole == "0") {
```

```
    await swal.fireAsync("Error", "You must select a role", SweetAlertIcon.Error);
    return;
}

var roleDTO = new EditRoleDTO() { Role = selectedRole, UserID = UserID };
var httpResponse = await repository.Post<EditRoleDTO>(url, roleDTO);

if(httpResponse.Error) {
    var ErrMessage = await httpResponse.GetErrMsg();
    await swal.fireAsync("Error", ErrMessage, SweetAlertIcon.Error);
}
else {
    await swal.fireAsync("Success", "Role was edited successfully", SweetAlertIcon.Success);
}

private async Task AssignRole() {
    await EditRole("api/users/assignRole");
}

private async Task RemoveRole() {
    await EditRole("api/users/removeRole");
}
```

También necesitamos traer los roles del usuario logueado al armar nuestro token. Para ello, agregamos el código en la clase **AccountsController.cs**. Transformamos el método **BuildToken** en una tarea asincrónica y modificamos las llamadas a ésta para que tenga su correspondiente **await**.

AccountsController.cs

```
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/accounts")]
    public class AccountsController: ControllerBase {
        private readonly UserManager<IdentityUser> userManager;
        private readonly SignInManager<IdentityUser> signInManager;
        private readonly IConfiguration configuration;

        public AccountsController(UserManager<IdentityUser> userManager,
            SignInManager<IdentityUser> signInManager,
```

```
IConfiguration configuration) {
    this.userManager = userManager;
    this.signInManager = signInManager;
    this.configuration = configuration;
}

[HttpPost("create")]
public async Task<ActionResult<UserTokenDTO>> CreateUser([FromBody] UserInfoDTO model) {
    var user = new IdentityUser { UserName = model.Email, Email = model.Email };
    var result = await userManager.CreateAsync(user, model.Password);

    if(result.Succeeded) {
        return await BuildToken(model);
    }
    else
        return BadRequest(result.Errors.First());
}

[HttpPost("login")]
public async Task<ActionResult<UserTokenDTO>> Login([FromBody] UserInfoDTO model) {
    //isPersistent: lo guarda en cookies
    //lockoutOnFailure: se lockea el usuario después varios intentos incorrectos.
    var result = await signInManager.PasswordSignInAsync(model.Email, model.Password,
    isPersistent: false, lockoutOnFailure: false);

    if(result.Succeeded)
        return await BuildToken(model);
    else
        return BadRequest("User/Password incorrect");
}

private async Task<UserTokenDTO> BuildToken(UserInfoDTO userInfo) {
    var claims = new List<Claim>() {
        //Esta info es accesible desde el frontend de Blazor. NO VA NINGÚN DATO SENSIBLE (clave,
        tarjetas de crédito, etc)
        new Claim(ClaimTypes.Name, userInfo.Email),
        new Claim("miValor", "Lo qu necesite...")
    };

    var user = await userManager.FindByEmailAsync(userInfo.Email);
    var roles = await userManager.GetRolesAsync(user!);

    foreach(var role in roles) {
        claims.Add(new Claim(ClaimTypes.Role, role));
    }

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["jwtkey"]!));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var expiration = DateTime.UtcNow.AddHours(1);
}
```

```

var token = new JwtSecurityToken(
    issuer: null,
    audience: null,
    claims: claims,
    expires: expiration,
    signingCredentials: creds
);

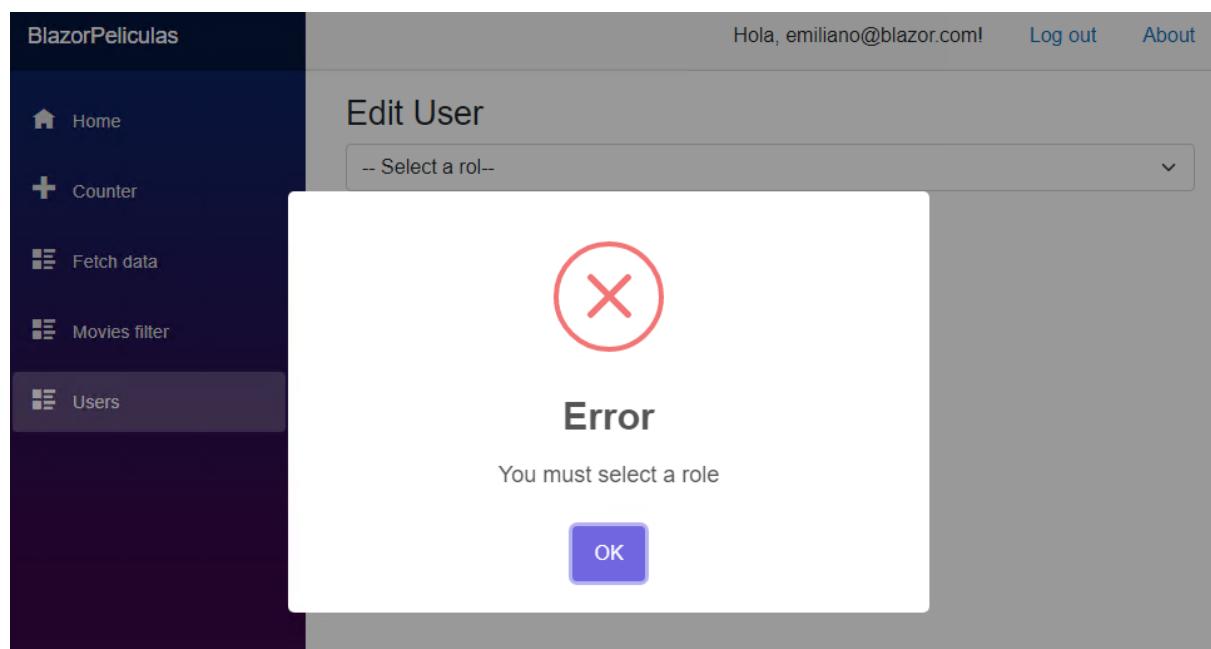
return new UserTokenDTO {
    Token = new JwtSecurityTokenHandler().WriteToken(token),
    Expiration = expiration
};
}
}
}
}

```

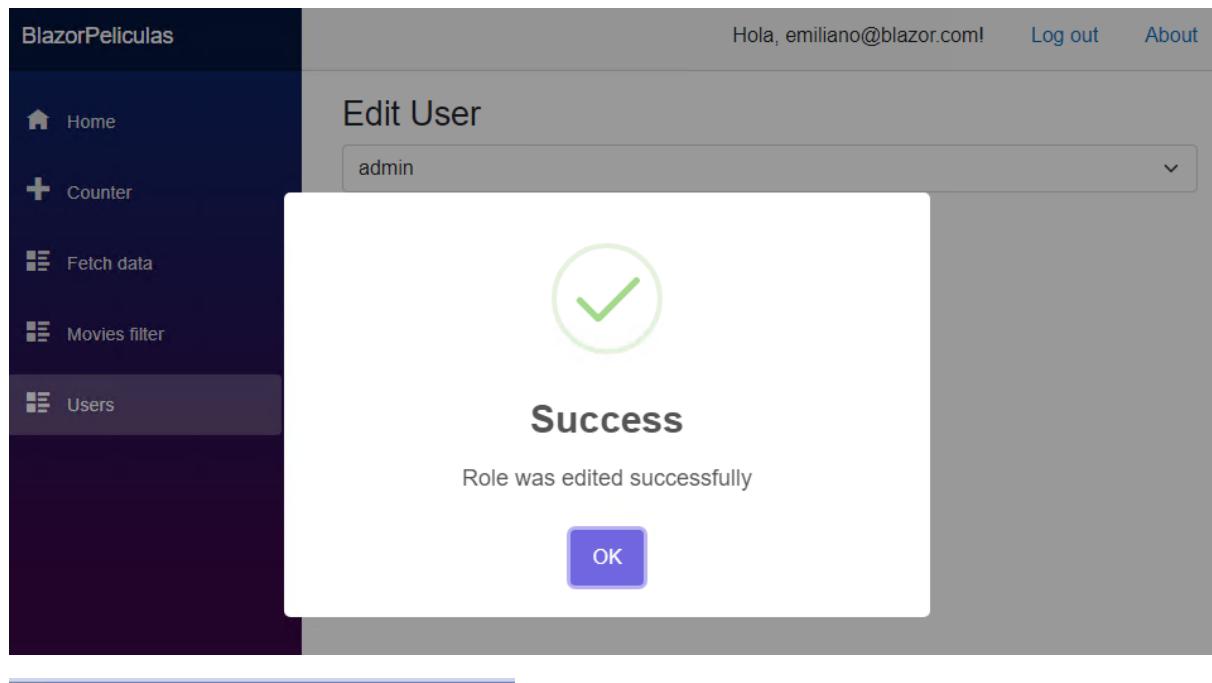
Veamos la tabla de UserRoles para constatar que no hay ninguno aún:

dbo.AspNetUserRoles [Data] ➔ X AccountsController.cs		
	User Id	Role Id
*	NULL	NULL

Ya podemos probar. Si intentamos agregar un rol sin elegir ninguno:

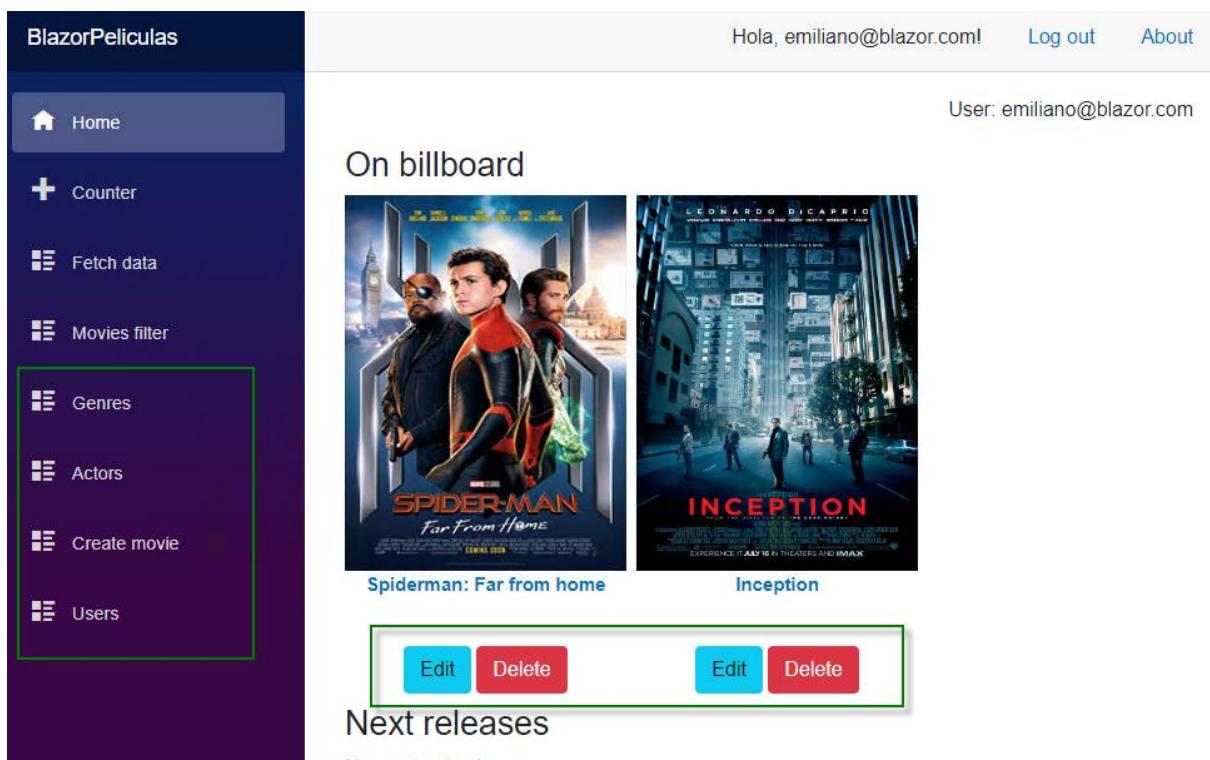


Si elegimos un rol y le damos a Assign Role recibimos el mensaje de éxito y vemos que la tabla ya tiene un registro:



dbo.AspNetUserRoles [Data] X		
	User Id	Role Id
▶	eb3-d05f5f424259	505630f9-a640-...
*	NULL	NULL

Si nos deslogueamos y volvemos a loguear, vemos que volvemos a tener todas las opciones de administrador (agregar géneros, actores, películas, etc):



Eliminamos el AuthorizeView de Index.razor que era una prueba.

```
Index.razor
@page "/"
@inject IRepository repository

<PageTitle>Blazor Movies</PageTitle>

<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
```

```

<h3>Next releases</h3>
<div>
    <MoviesList Movies="NextReleases">
        <Loading>
            
        </Loading>
        <NoRecords>
            <p>No movies to show</p>
        </NoRecords>
    </MoviesList>
</div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
        if(responseHTTP httpResponseMessage.IsSuccessStatusCode) {
            OnBoard = responseHTTP.Response!.OnBoard;
            NextReleases = responseHTTP.Response!.NextReleases;
        }
        else
            Console.WriteLine(responseHTTP httpResponseMessage.StatusCode);
    }
}
    
```

Y movemos el menú de entrada a Usuarios del menú izquierdo dentro de las opciones de administración:

```

Logout.razor
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">BlazorPelículas</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
    </nav>
</div>
    
```

```

</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="counter">
        <span class="oi oi-plus" aria-hidden="true"></span> Counter
    </NavLink>
</div>
<div class="nav-item px-3">
    <NavLink class="nav-link" href="fetchdata">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
    </NavLink>
</div>

<div class="nav-item px-3">
    <NavLink class="nav-link" href="movies/filter">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Movies filter
    </NavLink>
</div>

<AuthorizeView Roles="admin">
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="genres">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="actors">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/create">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
        </NavLink>
    </div>
</AuthorizeView>

</nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;
}

```

```
private void ToggleNavMenu()
{
    collapseNavMenu = !collapseNavMenu;
}
```

Para segurizar las APIs, además de exigir que un usuario esté logueado, pediremos que debe tener rol de administrador (por las dudas):

ActorsController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/actors"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Roles = "admin")]

    public class ActorsController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "people";

        public ActorsController(ApplicationDbContext context, IFileSaver fileSaver, IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Actor>>> Get([FromQuery] PaginationDTO pagination) {
            //return await context.Actors.ToListAsync();
            var queryable = context.Actors.AsQueryable();
            await HttpContext.InsertPaginationParametersInResponse(queryable, pagination.RecordCount);
            return await queryable.OrderBy(x => x.Name).ToPage(pagination).ToListAsync();
        }

        [HttpGet("search/{searchText}")]
        public async Task<ActionResult<IEnumerable<Actor>>> Get(string searchText) {
            if(string.IsNullOrWhiteSpace(searchText))
                return new List<Actor>();
        }
    }
}
```

```
searchText = searchText.ToLower();
return await context.Actors
    .Where(x => x.Name.ToLower().Contains(searchText))
    .Take(5)
    .ToListAsync();
}

[HttpGet("{id:int}")]
public async Task<ActionResult<Actor>> Get(int id) {
    var actor = await context.Actors.FirstOrDefaultAsync(actor => actor.ID == id);

    if(actor is null)
        return NotFound();

    return actor;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Actor actor) {
    if(!string.IsNullOrEmpty(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actor.Photo = await fileSaver.SaveFile(photoActor, ".jpg", container);
    }

    context.Add(actor);
    await context.SaveChangesAsync();
    return actor.ID;
}

[HttpPut]
public async Task<ActionResult> Put(Actor actor) {
    var actorDB = await context.Actors.FirstOrDefaultAsync(a => a.ID == actor.ID);

    if(actorDB is null)
        return NotFound();
    //Tomá las propiedades de actor y pasalas a actorDB
    actorDB = mapper.Map(actor, actorDB);

    if(!string.IsNullOrEmpty(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actorDB.Photo = await fileSaver.EditFile(photoActor, ".jpg", container, actorDB.Photo!);
    }

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}
```

```
[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var actor = await context.Actors.FirstOrDefaultAsync(x => x.ID == id);

    if(actor is null)
        return NotFound();

    context.Remove(actor);    //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if(!string.IsNullOrWhiteSpace(actor.Photo))
        await fileSaver.DeleteFile(actor.Photo!, container);

    return NoContent();
}
}
```

GenresController.cs

```
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Identity.Client;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/genres"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Roles = "admin")]
    public class GenresController : ControllerBase {
        private readonly ApplicationDbContext context;
        public GenresController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpGet, AllowAnonymous]
        public async Task<ActionResult<IEnumerable<Genre>>> Get() {
            return await context.Genres.ToListAsync();
        }

        [HttpGet("{id:int}")]
        public async Task<ActionResult<Genre>> Get(int id) {
            var genre = await context.Genres.FirstOrDefaultAsync(genre => genre.ID == id);

            if (genre is null)
                return NotFound();

            return genre;
        }
    }
}
```

```
}
```

```
[HttpPost]
public async Task<ActionResult<int>> Post(Genre genre) {
    context.Add(genre);
    await context.SaveChangesAsync();
    return genre.ID;
}

[HttpPut]
public async Task<ActionResult<int>> Put(Genre genre) {
    context.Update(genre); //Marco el género para ser actualizado.
    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent(); //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var affectedFiles = await context.Genres
        .Where(x => x.ID == id)
        .ExecuteDeleteAsync();

    if(affectedFiles == 0)
        return NotFound();

    return NoContent();
}
}
```

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;
```

```
namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Roles = "admin")]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
```

```
private readonly UserManager<IdentityUser> userManager;
private readonly string container = "movies";

public MoviesController(ApplicationDbContext context,
    IFileSaver fileSaver,
    IMapper mapper,
    UserManager<IdentityUser> userManager) {
    this.context = context;
    this.fileSaver = fileSaver;
    this.mapper = mapper;
    this.userManager = userManager;
}

[HttpGet, AllowAnonymous]
public async Task<ActionResult<HomePageDTO>> Get() {
    var limit = 6;
    var onBoardMovies = await context.Movies
        .Where(movie => movie.OnBillboard)
        .Take(limit)
        .OrderByDescending(movie => movie.ReleaseDate)
        .ToListAsync();
    var today = DateTime.Today;
    var nextReleases = await context.Movies
        .Where(movie => movie.ReleaseDate > today)
        .Take(limit)
        .OrderBy(movie => movie.ReleaseDate)
        .ToListAsync();

    var result = new HomePageDTO {
        OnBoard = onBoardMovies,
        NextReleases = nextReleases
    };
    return result;
}

[HttpGet("{id:int}"), AllowAnonymous]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }
}
```

```
var votesMedia = 0.0;
var userVote = 0;

if(await context.VotesMovies.AnyAsync(x => x.MovieID == id)) {
    //Alguien ha votado por la película
    votesMedia = await context.VotesMovies
        .Where(x => x.MovieID == id)
        .AverageAsync(x => x.Voto);

    if(HttpContext.User.Identity!.IsAuthenticated) {
        var user = await userManager.FindByEmailAsync(HttpContext.User.Identity!.Name!);

        if(user is null) {
            //No debería pasar nunca, pero por las dudas...
            return BadRequest("User was not found");
        }

        var userID = user.Id;

        var userVoteDB = await context.VotesMovies
            .FirstOrDefaultAsync(x => x.MovieID == id && x.UserID == userID);

        if(userVoteDB is not null)
            userVote = userVoteDB.Voto;
    }
}

var model = new MovieViewDTO();
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("filter"), AllowAnonymous]
public async Task<ActionResult<List<Movie>>> Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
```

```
.Where(x => x.Title.Contains(model.Title));

if(model.Onbillboard)
    queryableMovies = queryableMovies
        .Where(x => x.OnBillboard);

if(model.Releases) {
    var today = DateTime.Today;

    queryableMovies = queryableMovies
        .Where(x => x.ReleaseDate >= today);
}

if (model.GenreID != 0)
    queryableMovies = queryableMovies
        .Where(x => x.GenresMovie
            .Select(y => y.GenreID)
            .Contains(model.GenreID));

if(model.MostVoted) {
    queryableMovies = queryableMovies.OrderByDescending(m => m.VotesMovies.Average(vm
=> vm.Voto));
}

await HttpContext.InserPaginationParametersInResponse(queryableMovies,
model.RecordCount);

//Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
```

```
model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, ".jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPut]
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
    return NoContent();           //Todo se hizo correctamente
}
```

```
}
```



```
[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if (movie is null)
        return NotFound();

    context.Remove(movie);    //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if (!string.IsNullOrWhiteSpace(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

    return NoContent();      //Todo se hizo correctamente
}
```

UsersController.cs

```
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/users"), Authorize(AuthenticationSchemes =
JwtBearerDefaults.AuthenticationScheme, Roles = "admin")]
    public class UsersController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly UserManager<IdentityUser> userManager;

        public UsersController(ApplicationDbContext context,
            UserManager<IdentityUser> userManager) {
            this.context = context;
            this.userManager = userManager;
        }

        [HttpGet]
        public async Task<ActionResult<List<UserDTO>>> Get([FromQuery] PaginationDTO pagination) {
            var queryable = context.Users.AsQueryable();
            await HttpContext.InferPaginationParametersInResponse(queryable,
                pagination.RecordCount);
            return await queryable.ToPage(pagination)
                .Select(x => new UserDTO { ID = x.Id, Email = x.Email! });
        }
    }
}
```

```
.ToListAsync();
}

[HttpGet("roles")]
public async Task<ActionResult<List<RoleDTO>>> Get() {
    return await context.Roles.Select(x => new RoleDTO { Name = x.Name }).ToListAsync();
}

[HttpPost("assignRole")]
public async Task<ActionResult> AssignRoleToUser(EditRoleDTO editRoleDTO) {
    var user = await userManager.FindByIdAsync(editRoleDTO.UserID);
    if(user is null)
        return BadRequest("User was not found!");

    await userManager.AddToRoleAsync(user, editRoleDTO.Role);
    return NoContent();
}

[HttpPost("removeRole")]
public async Task<ActionResult> RemoveRoleFromUser(EditRoleDTO editRoleDTO) {
    var user = await userManager.FindByIdAsync(editRoleDTO.UserID);
    if(user is null)
        return BadRequest("User was not found!");

    await userManager.RemoveFromRoleAsync(user, editRoleDTO.Role);
    return NoContent();
}
}
```

Renovando el JWT

En algunos casos, preferiremos que el token no tenga una expiración de 1 año o 1 hora. En aplicaciones como un banco habría que poner que el token sea de pocos minutos. Pero, tampoco se pretendería que el usuario tenga que estar cada 5 minutos volviéndose a loguear sino que la expiración vaya actualizándose conforme a su uso.

Crearemos un endpoint nuevo para poder renovar el token que estará segurizado para que sólo pueda ser usado por alguien que tenga token.

AccountsController.cs

```
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.IdentityModel.Tokens;
```

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/accounts")]
    public class AccountsController: ControllerBase {
        private readonly UserManager<IdentityUser> userManager;
        private readonly SignInManager<IdentityUser> signInManager;
        private readonly IConfiguration configuration;

        public AccountsController(UserManager<IdentityUser> userManager,
            SignInManager<IdentityUser> signInManager,
            IConfiguration configuration) {
            this.userManager = userManager;
            this.signInManager = signInManager;
            this.configuration = configuration;
        }

        [HttpPost("create")]
        public async Task<ActionResult<UserTokenDTO>> CreateUser([FromBody] UserInfoDTO model) {
            var user = new IdentityUser { UserName = model.Email, Email = model.Email };
            var result = await userManager.CreateAsync(user, model.Password);

            if(result.Succeeded) {
                return await BuildToken(model);
            }
            else
                return BadRequest(result.Errors.First());
        }

        [HttpPost("login")]
        public async Task<ActionResult<UserTokenDTO>> Login([FromBody] UserInfoDTO model) {
            //isPersistent: lo guarda en cookies
            //lockoutOnFailure: se lockea el usuario después varios intentos incorrectos.
            var result = await signInManager.PasswordSignInAsync(model.Email, model.Password,
isPersistent: false, lockoutOnFailure: false);

            if(result.Succeeded)
                return await BuildToken(model);
            else
                return BadRequest("User/Password incorrect");
        }

        [HttpGet("renewToken")]
        [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
        public async Task<ActionResult<UserTokenDTO>> Renew() {
            //Construimos un userInfo para poder llamar a BuildToken que recibe uno como parámetro.
            var userInfo = new UserInfoDTO() {
                Email = HttpContext.User.Identity!.Name!
            }
        }
    }
}
```

```
    }

    return await BuildToken(userInfo);
}

private async Task<UserTokenDTO> BuildToken(UserInfoDTO userInfo) {
    var claims = new List<Claim>() {
        //Esta info es accesible desde el frontend de Blazor. NO VA NINGÚN DATO SENSIBLE (clave,
        tarjetas de crédito, etc)
        new Claim(ClaimTypes.Name, userInfo.Email),
        new Claim("miValor", "Lo qu necesite...")
    };

    var user = await userManager.FindByEmailAsync(userInfo.Email);
    var roles = await userManager.GetRolesAsync(user!);

    foreach(var role in roles) {
        claims.Add(new Claim(ClaimTypes.Role, role));
    }

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(configuration["jwtkey"]!));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    var expiration = DateTime.UtcNow.AddHours(1);

    var token = new JwtSecurityToken(
        issuer: null,
        audience: null,
        claims: claims,
        expires: expiration,
        signingCredentials: creds
    );

    return new UserTokenDTO {
        Token = new JwtSecurityTokenHandler().WriteToken(token),
        Expiration = expiration
    };
}
}
```

En la clase **JWTAuthProvider.cs** agregaremos un campo para guardar en el **LocalStorage** la fecha de expiración. Necesitaremos un método auxiliar que nos permita limpiar la data relacionada a los tokens y también necesitamos guardar el tiempo de expiración del token.

En el método **Login** cambiamos el parámetro recibido por uno de tipo **UserTokenDTO** que nos permita tener tanto el token como la fecha de expiración.

JWTAuthProvider.cs

```
using BlazorPeliculas.Client.Helpers;
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.JSInterop;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http.Headers;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class JWTAuthProvider : AuthenticationStateProvider, ILoginService {
        private readonly IJSRuntime js;
        private readonly HttpClient httpClient;

        public JWTAuthProvider(IJSRuntime js, HttpClient httpClient) {
            this.js = js;
            this.httpClient = httpClient;
        }

        public static readonly string TOKENKEY = "TOKENKEY";
        public static readonly string EXPIRATIONTOKENKEY = "EXPIRATIONTOKENKEY";

        private AuthenticationState anonymous = new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity()));
        public async override Task<AuthenticationState> GetAuthenticationStateAsync() {
            var token = await js.GetFromLocalStorage(TOKENKEY);

            if(token is null) {
                //Es un usuario anónimo.
                return anonymous;
            }

            return ConstructAuthenticationState(token.ToString()!);
        }

        private AuthenticationState ConstructAuthenticationState(string token) {
            httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
            token);
            var claims = ParseClaimsOutOfJWT(token);
            return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));
        }

        private IEnumerable<Claim> ParseClaimsOutOfJWT(string token) {
            var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
            var deserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

            return deserializedToken.Claims;
        }

        public async Task Login(UserTokenDTO tokenDTO) {
```

```
await js.SetInLocalStorage(TOKENKEY, tokenDTO.Token);
await js.SetInLocalStorage(EXPIRATIONTOKENKEY, tokenDTO.Expiration.ToString());
var authState = ConstructAuthenticationState(tokenDTO.Token);
NotifyAuthenticationStateChanged(Task.FromResult(authState)); //Le notifico a Blazor que
cambió el estado.

}

public async Task Logout() {
    await Clean();
    httpClient.DefaultRequestHeaders.Authorization = null;
    NotifyAuthenticationStateChanged(Task.FromResult(anonymous)); //Le notifico a Blazor que
cambió el estado.
}

private async Task Clean() {
    await js.RemoveFromLocalStorage(TOKENKEY);
    await js.RemoveFromLocalStorage(EXPIRATIONTOKENKEY);
    httpClient.DefaultRequestHeaders.Authorization = null;
}
}
```

La interfaz **ILoginService.cs** debe modificarse para que sea coherente con los cambios recientes:

```
ILoginService.cs
using BlazorPeliculas.Shared.DTOs;

namespace BlazorPeliculas.Client.Auth {
    public interface ILoginService {
        Task Login(UserTokenDTO tokenDTO);
        Task Logout();
    }
}
```

Mientras que los componentes Register.razor y Login.razor deben ser modificados para que pase el parámetro correctamente. En ambos casos no hay que pasar el **Response!.Token!** sino simplemente **Response!**:

```
Register.razor
@page "/Register"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal
@inject ILoginService loginService

<h3>Register</h3>
```

```

<EditForm Model="userInfo" OnValidSubmit="CreateUser">
    <DataAnnotationsValidator />
    <div class="mb-3">
        <label>Email:</label>
        <div>
            <InputText class="form-control" @bind-Value="userInfo.Email" />
            <ValidationMessage For="@(() => userInfo.Email)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Password:</label>
        <div>
            <InputText type="text" class="form-control" @bind-Value="userInfo.Password" />
            <ValidationMessage For="@(() => userInfo.Password)" />
        </div>
    </div>

    <button type="submit" class="btn btn-primary">Register</button>
</EditForm>

@code {
    private UserInfoDTO userInfo = new UserInfoDTO();

    private async Task CreateUser() {
        var responseHTTP = await repository.Post<UserInfoDTO, UserTokenDTO>("api/accounts/create",
            userInfo);

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            await loginService.Login(responseHTTP.Response!);
            navManager.NavigateTo("");
        }
    }
}

```

Login.razor

```

@page "/Login"
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal
@inject ILoginService loginService

<h3>Login</h3>
<EditForm Model="userInfo" OnValidSubmit="LoginUser">
    <DataAnnotationsValidator />

```

```
<div class="mb-3">
    <label>Email:</label>
    <div>
        <InputText class="form-control" @bind-Value="userInfo.Email" />
        <ValidationMessage For="@(() => userInfo.Email)" />
    </div>
</div>

<div class="mb-3">
    <label>Password:</label>
    <div>
        <InputText type="password" class="form-control" @bind-Value="userInfo.Password" />
        <ValidationMessage For="@(() => userInfo.Password)" />
    </div>
</div>

<button type="submit" class="btn btn-primary">Login</button>
</EditForm>

@code {
    private UserInfoDTO userInfo = new UserInfoDTO();

    private async Task LoginUser() {
        var responseHTTP = await repository.Post<UserInfoDTO, UserTokenDTO>("api/accounts/login",
userInfo);

        if(responseHTTP.Error) {
            var ErrMessage = await responseHTTP.GetErrMsg();
            await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
        }
        else {
            await loginService.Login(responseHTTP.Response!);
            navManager.NavigateTo("");
        }
    }
}
```

En la clase **JWTAuthProvider.cs** revisaremos que exista la fecha de expiración en el LocalStorage y si no la tuviera, contestaría que es un usuario anónimo. También, revisará la fecha de expiración. Si ya expiró, contestaría que es anónimo.

Por otro lado, sólo renovaremos el token si el mismo está próximo a expirar (dentro de los 5 minutos). Inyectamos el repositorio porque necesitamos poder hacer una petición al endpoint que creamos para renovar el token.

Creamos el método **ManageTokenRenewal** que nos permitirá renovar periódicamente el token. Hacemos un Pull para incluir la firma en la interfaz.

JWTAuthProvider.cs

```
using BlazorPeliculas.Client.Helpers;
using BlazorPeliculas.Client.Repositories;
using BlazorPeliculas.Shared.DTOs;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.JSInterop;
using System.IdentityModel.Tokens.Jwt;
using System.Net.Http.Headers;
using System.Security.Claims;

namespace BlazorPeliculas.Client.Auth {
    public class JWTAuthProvider : AuthenticationStateProvider, ILoginService {
        private readonly IJSRuntime js;
        private readonly HttpClient httpClient;
        private readonly IRepository repository;

        public JWTAuthProvider(IJSRuntime js,
            HttpClient httpClient,
            IRepository repository) {
            this.js = js;
            this.httpClient = httpClient;
            this.repository = repository;
        }

        public static readonly string TOKENKEY = "TOKENKEY";
        public static readonly string EXPIRATIONTOKENKEY = "EXPIRATIONTOKENKEY";

        private AuthenticationState anonymous = new AuthenticationState(new ClaimsPrincipal(new
ClaimsIdentity()));

        public async override Task<AuthenticationState> GetAuthenticationStateAsync() {
            var token = await js.GetFromLocalStorage(TOKENKEY);

            if(token is null) {
                //Es un usuario anónimo.
                return anonymous;
            }

            DateTime expirationTime;
            var expirationTimeObject = await js.GetFromLocalStorage(EXPIRATIONTOKENKEY);

            if(expirationTimeObject is null) {
                //No tiene tiempo de expiración
                await Clean();
                return anonymous;
            }

            if(DateTime.TryParse(expirationTimeObject.ToString(), out expirationTime)) {
                if(expiredToken(expirationTime)) {
                    //El token expiró
                    await Clean();
                }
            }
        }
    }
}
```

```

        return anonymous;
    }

    if(MustRenewToken(expirationTime)) {
        token = await RenewToken(token.ToString()!);
    }

    return ConstructAuthenticationState(token.ToString()!);
}

private bool expiredToken(DateTime expirationTime) {
    return expirationTime <= DateTime.UtcNow;
}

private bool MustRenewToken(DateTime expirationTime) {
    return expirationTime.Subtract(DateTime.UtcNow) < TimeSpan.FromMinutes(5);
}

public async Task ManageTokenRenewal() {
    DateTime expirationTime;
    var expirationTimeObject = await js.GetFromLocalStorage(EXPIRATIONTOKENKEY);

    if(DateTime.TryParse(expirationTimeObject.ToString(), out expirationTime)) {
        if(expiredToken(expirationTime)) {
            //El token expiró
            await Logout();
        }

        if(MustRenewToken(expirationTime)) {
            var token = await js.GetFromLocalStorage(TOKENKEY);
            var newToken = await RenewToken(token.ToString()!);
            var authState = ConstructAuthenticationState(newToken);

            //Debo notificar al cliente porque puedo tener nuevos roles en el token.
            NotifyAuthenticationStateChanged(Task.FromResult(authState));
        }
    }
}

private async Task<string> RenewToken(string token) {
    Console.WriteLine("Renewing the token...");
    //Nos aseguramos de que el httpClient tenga el token.
    httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
    token);

    var newTokenResponse = await repository.Get<UserTokenDTO>("api/accounts/RenewToken");
    var newToken = newTokenResponse.Response!;

    await js.SetInLocalStorage(TOKENKEY, newToken.Token);
    await js.SetInLocalStorage(EXPIRATIONTOKENKEY, newToken.Expiration.ToString());
}

```

```
        return newToken.Token;
    }

    private AuthenticationState ConstructAuthenticationState(string token) {
        httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("bearer",
token);
        var claims = ParseClaimsOutOfJWT(token);
        return new AuthenticationState(new ClaimsPrincipal(new ClaimsIdentity(claims, "jwt")));
    }

    private IEnumerable<Claim> ParseClaimsOutOfJWT(string token) {
        var jwtSecurityTokenHandler = new JwtSecurityTokenHandler();
        var deserializedToken = jwtSecurityTokenHandler.ReadJwtToken(token);

        return deserializedToken.Claims;
    }

    public async Task Login(UserTokenDTO tokenDTO) {
        await js.SetInLocalStorage(TOKENKEY, tokenDTO.Token);
        await js.SetInLocalStorage(EXPIRATIONTOKENKEY, tokenDTO.Expiration.ToString());
        var authState = ConstructAuthenticationState(tokenDTO.Token);
        NotifyAuthenticationStateChanged(Task.FromResult(authState)); //Le notifico a Blazor que
cambió el estado.
    }

    public async Task Logout() {
        await Clean();
        httpClient.DefaultRequestHeaders.Authorization = null;
        NotifyAuthenticationStateChanged(Task.FromResult(anonymous)); //Le notifico a Blazor que
cambió el estado.
    }

    private async Task Clean() {
        await js.RemoveFromLocalStorage(TOKENKEY);
        await js.RemoveFromLocalStorage(EXPIRATIONTOKENKEY);
        httpClient.DefaultRequestHeaders.Authorization = null;
    }
}
```

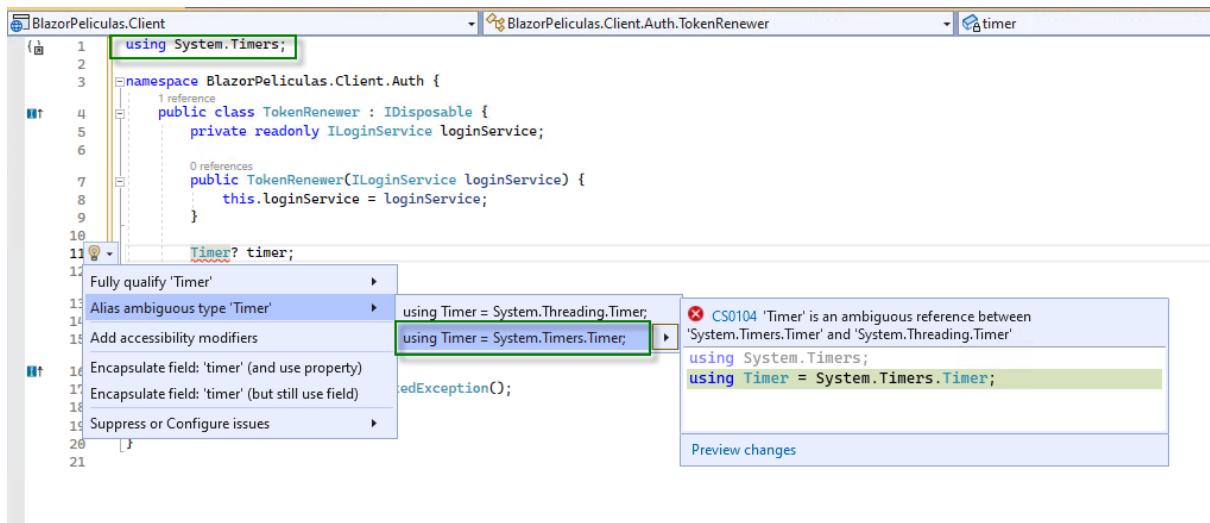
ILoginService.cs

```
using BlazorPeliculas.Shared.DTOs;

namespace BlazorPeliculas.Client.Auth {
    public interface ILoginService {
        Task Login(UserTokenDTO tokenDTO);
        Task Logout();
        Task ManageTokenRenewal();
    }
}
```

Necesitamos un código que nos permita revisar cada X tiempo si un token debe ser renovado. Crearemos la clase **TokenRenewer.cs** en la carpeta **Auth** del proyecto **Client** e implementaremos de **IDisposable**.

Con en .NET hay una colisión de nombres con respecto a Timers, es necesario agregar manualmente el using y resolver la ambigüedad con **Ctrl+.**:



TokenRenewer.cs

```
using System.Timers;
using Timer = System.Timers.Timer;

namespace BlazorPeliculas.Client.Auth {
    public class TokenRenewer : IDisposable {
        private readonly ILoginService loginService;

        public TokenRenewer(ILoginService loginService) {
            this.loginService = loginService;
        }

        Timer? timer;

        public void Start() {
            timer = new Timer();
            timer.Interval = 1000 * 60 * 4; // 4 minutos (debe ser menor al umbral definido (5 minutos)
            timer.Elapsed += Timer_Elapsed;
            timer.Start();
        }

        private void Timer_Elapsed(object? sender, ElapsedEventArgs e) {
            loginService.ManageTokenRenewal();
        }

        public void Dispose() {
    }
```

```
//Tenemos que limpiar el timer ya que utiliza recursos no manejados que tenemos que limpiar.  
timer?.Dispose();  
}  
}  
}
```

Debemos agregar este servicio en la clase **Program.cs** del proyecto **Client**.

Program.cs

```
using BlazorPeliculas.Client;  
using BlazorPeliculas.Client.Auth;  
using BlazorPeliculas.Client.Repositories;  
using CurrieTechnologies.Razor.SweetAlert2;  
using Microsoft.AspNetCore.Components.Authorization;  
using Microsoft.AspNetCore.Components.Web;  
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;  
using Microsoft.Extensions.DependencyInjection;  
  
var builder = WebAssemblyHostBuilder.CreateDefault(args);  
builder.RootComponents.Add<App>("#app");  
builder.RootComponents.Add<HeadOutlet>("head::after");  
  
builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new  
Uri(builder.HostEnvironment.BaseAddress) });  
configureServices(builder.Services);  
await builder.Build().RunAsync();  
  
void configureServices(IServiceCollection services) {  
    services.AddScoped< IRepository, Repository>(); /* Inyectar un IRepository.  
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la  
    clase Repository. */  
    services.AddSweetAlert2();  
    services.AddAuthorizationCore();  
  
    services.AddScoped<JWTAuthProvider>();  
    services.AddScoped<AuthenticationStateProvider, JWTAuthProvider>(provider =>  
        provider.GetService<JWTAuthProvider>());  
  
    services.AddScoped<ILoginService, JWTAuthProvider>(provider =>  
        provider.GetService<JWTAuthProvider>());  
  
    services.AddScoped< TokenRenewer>();  
}
```

En el componente MainLayout.razor inyectamos este servicio y lo iniciamos:

MainLayout.razor

```
@inherits LayoutComponentBase
```

```

@inject BlazorPeliculas.Client.Auth.TokenRenewer tokenRenewer

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <AuthLinks />
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>

@code {
    protected override void OnInitialized() {
        tokenRenewer.Start();
    }
}

```

Al ejecutar la aplicación, aparecemos deslogueados porque el código detecta que no tenemos EXPIRATIONTOKENKEY en el LocalStorage y nos declara anónimo:

```

  28
  29
  30
  31
  32
  33
  34
  35
  36
  37
  38
  39
  40
  41
  42
  43

```

```

    public async override Task<AuthenticationState> GetAuthenticationStateAsync() {
        var token = await js.GetFromLocalStorage(TOKENKEY);

        if(token is null) {
            //Es un usuario anónimo.
            return anonymous;
        }

        DateTime expirationTime;
        var expirationTimeObject = await js.GetFromLocalStorage(EXPIRATIONTOKENKEY);

        if(expirationTimeObject is null) {
            //No tiene tiempo de expiración
            await Clean();
            return anonymous;
        }
    }

```

BlazorPelículas

Home Counter Fetch data Movies filter

On billboard

Spiderman: Far from home Inception

Next releases

No movies to show

Si nos logueamos y presionamos F12 podemos ir a Application y ver que en **LocalStorage** está tanto el **TOKENKEY** como el **EXPIRATIONTOKENKEY**:

BlazorPelículas

Hola, emiliano@blazor.com! Log out About

On billboard

Application

Key	Value
TOKENKEY	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hcy5...en-GB
EXPIRATIONTOKENKEY	17/02/2023 16:43:09

Storage

- Manifest
- Service Workers
- Storage
- Local Storage (https://localhost:7152)
- Session Storage
- IndexedDB
- Web SQL
- Cookies
- Trust Tokens

Siendo las 12:43 del 17/02, se puede ver que la fecha de expiración es dentro de 1 hora (a las 13:43:09 GMT-3).

Podríamos cambiar la siguiente línea de **AccountsController.cs**:

Desde	A
<code>var expiration = DateTime.UtcNow.AddHours(1);</code>	<code>var expiration = DateTime.UtcNow.AddMinutes(1);</code>

Para que se inicialice el token con una fecha de expiración dentro de un minuto. Y la siguiente línea de **TokenRenewer.cs**:

Desde	A
timer.Interval = 1000 * 60 * 4;	timer.Interval = 1000 * 5;

Para que se actualice automáticamente cada 5 segundos. La siguiente prueba se inició a las 13:25:29 y en el **LocalStorage** se ve que el token caducará al minuto siguiente:

Key	Value
TOKENKEY	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hc... en-GB
i18nextLng	
EXPIRATIONTOKENKEY	17/02/2023 16:26:29

Sin embargo, se puede apreciar que el mismo va siendo actualizado cada 5 segundos:

Key	Value
TOKENKEY	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJodHRwOi8vc2NoZW1hc... en-GB
i18nextLng	
EXPIRATIONTOKENKEY	17/02/2023 16:26:49

Incluso, en la consola puede verse que ya se ingresó 8 veces en el método de renovación:

```

DenyAnonymousAuthorizationRequirement: Requires an authenticated user.
info: Microsoft.AspNetCore.Authorization.DefaultAuthorizationService[2]
      Authorization failed. These requirements were not met:
      RolesAuthorizationRequirement:User.IsInRole must be true for one of the following roles: (admin)
⑧ Renewing the token...
>

```

Deslogueo automático

Crearemos un timer del lado de javascript que se resetee automáticamente si el usuario realizará alguna acción. Mientras que si no realiza nada, el timer seguirá activo y al cumplirse el tiempo indicado se ejecutaría la función indicada (logout). Ésta invocaría a la función Logout de nuestra app ([.NET](#)).

Esto lo hacemos en el archivo **Utilities.js**.

Utilities.js
function DotNetStaticTest() {

```
DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
    .then(result => {
        console.log('Count from js: ' + result);
    });
}

function DotNetInstanceTest(dotNetHelper) {
    dotNetHelper.invokeMethodAsync("IncrementCount");
}

function inactiveTimer(dotnetHelper) {
    var timer;

    document.onmousemove = resetimer;
    document.onkeypress = resetimer;

    function resetimer() {
        clearTimeout(timer);
        timer = setTimeout(logout, 3 * 1000); // 3 segundos
    }
}

function logout() {
    dotnetHelper.invokeMethodAsync("Logout");
}
```

Lo hacemos a los 3 segundos para probarlo (y no tener que estar mucho tiempo sin hacer nada para ver si funciona).

En el componente **MainLayout.razor** inyectamos el IJSRuntime para ejecutar esa función recién creada y el NavigationManager para redirigir al usuario a la página de logout.

Agregamos un **CascadingParameter** del AuthenticationState que nos permitirá, entre otras cosas, determinar si el usuario está logueado o no desde código de C#.

A su vez, necesitamos cambiar al evento **OnInitializedAsync** para poder ejecutar la función de JS de manera asíncrona.

```
MainLayout.razor
@inherits LayoutComponentBase
@inject BlazorPeliculas.Client.Auth.TokenRenewer tokenRenewer
@inject IJSRuntime js
@inject NavigationManager navManager

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>
```

```
<main>
    <div class="top-row px-4">
        <AuthLinks />
        <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
    </div>

    <article class="content px-4">
        @Body
    </article>
</main>
</div>

@code {
    [CascadingParameter]
    public Task<AuthenticationState> AuthenticationStateTask { get; set; } = null;

    protected async override Task OnInitializedAsync() {
        await js.InvokeVoidAsync("inactiveTimer", DotNetObjectReference.Create(this));
        tokenRenewer.Start();
    }

    [JSInvokable]
    public async Task Logout() {
        var authState = await AuthenticationStateTask;
        if(authState.User.Identity!.IsAuthenticated)
            navManager.NavigateTo("/logout");
    }
}
```

Con esos cambios se puede ver que si no movemos el mouse ni presionamos ninguna tecla, el sistema nos redirige a la página de logout.

Resumen



Podemos realizar un sistema de usuarios con Identity y Entity Framework Core de una manera sencilla y segura



Los claims son informaciones de los usuarios las cuales podemos considerar como verdaderas



El componente AuthorizeView nos permite definir qué le va a salir al usuario según su estado de autenticación.



Authorize nos permite bloquear el acceso a un componente ruteable en caso de que el usuario no se encuentre autorizado



Los roles nos permiten definir categorías de usuarios, y con esas categorías podemos otorgar permisos

Despliegue

Lo primero que vamos a hacer es que vamos a instalar el **Entity Framework Core Cli**. Para eso ejecutamos el siguiente comando:

```
C:\Users\EFM>dotnet tool install --global dotnet-ef  
You can invoke the tool using the following command: dotnet-ef  
Tool 'dotnet-ef' (version '7.0.3') was successfully installed.
```

Esto es importante porque con esta herramienta es que vamos a poder correr las migraciones en nuestra base de datos de Azure. Claro está, como vamos a instalar una aplicación en un Azure App Service. Necesitas una cuenta de Azure.

Copiamos los seteos desde **appSettings.Development.json** en **appSettings.json**:

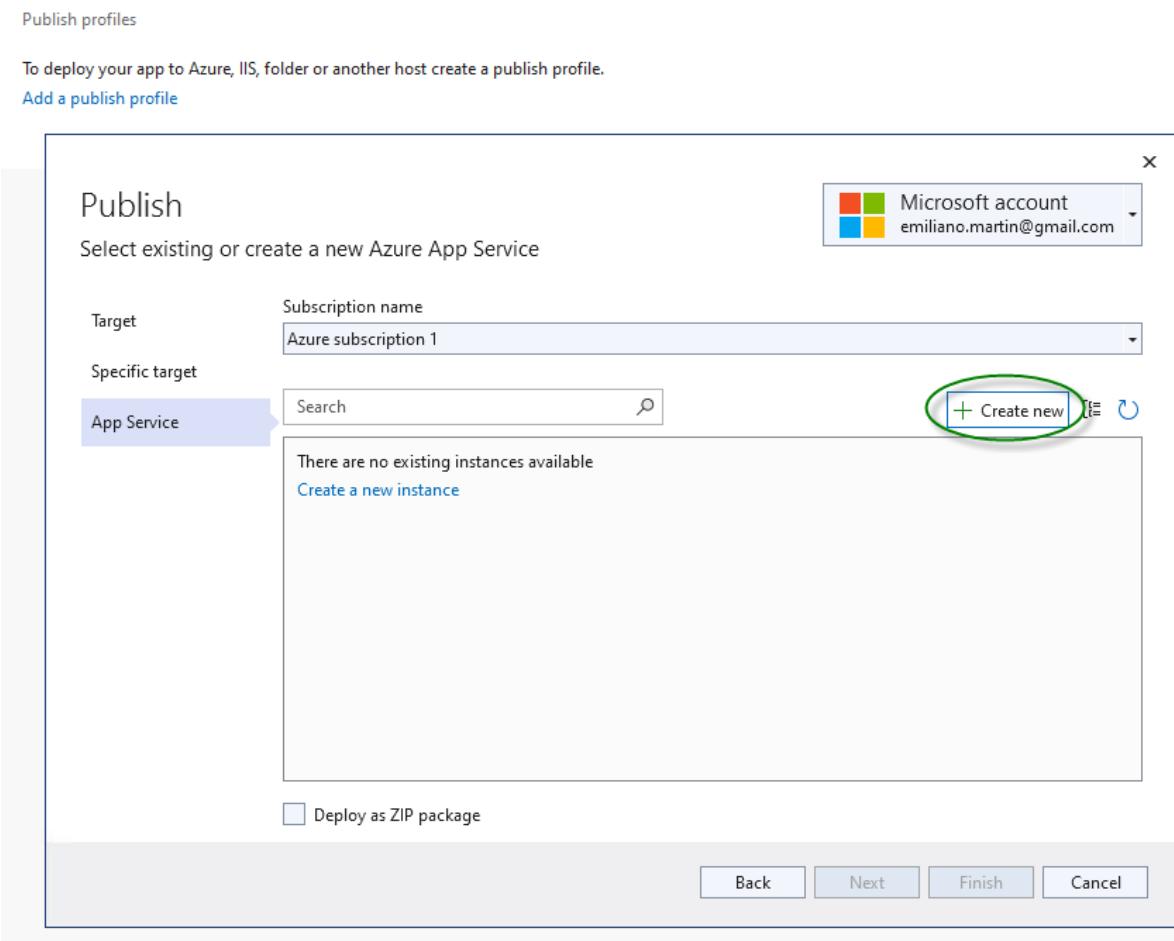
Logout.razor

```
{  
  "ConnectionStrings": {  
    "AzureStorage": "blablabla"  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*",  
  "jwtkey":  
    "utLEZPsC96sxppPqo5KtvI8Gm1VpoiGdVfdOZvcdpVgBcMbNLAxIZ7M9bsRwW1PUBTY05vBh5laA5bXw9MX  
    yJNUg3SOPDZWpCEh1587RZ2OOomLGTK0kPKLZJtFxR7k1"  
}
```

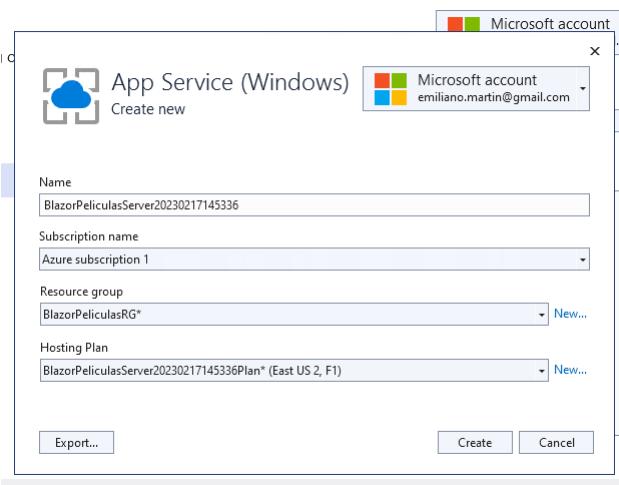
En AzureStorage dice blablabla porque las imágenes las estaba guardando localmente en la máquina en lugar de en Azure. No copiamos el **DefaultConnection** porque no utilizaremos una base local sino una de Azure.

Hacemos click-derecho en el proyecto **Server** y hacemos click en **Publish**. Elegimos **Azure**, luego en **Azure App Service (Windows)**.

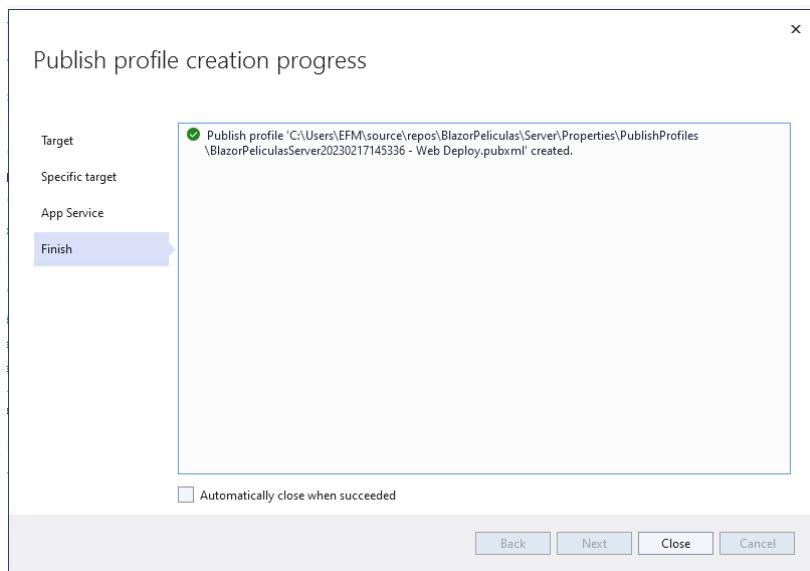
Nos logueamos (o creamos una cuenta de Azure nueva) y elegimos **Create new**:



Elegimos el nombre para nuestra appservice, la suscripción el resource group y el hosting plan. Para este último, elegí uno gratuito:



Presionamos **Create** y esperamos que se termine de crear. Cuando suceda, presionamos **Finish**. Al hacerlo, nos mostrará el path donde se creó el perfil de publicación:



Al cerrar la ventana, vemos el resumen del perfil creado:

No puedo continuar con el proceso porque no me deja agregar el servicio de dependencia de SQL Server (seguramente porque elegí un hosting plan gratuito).

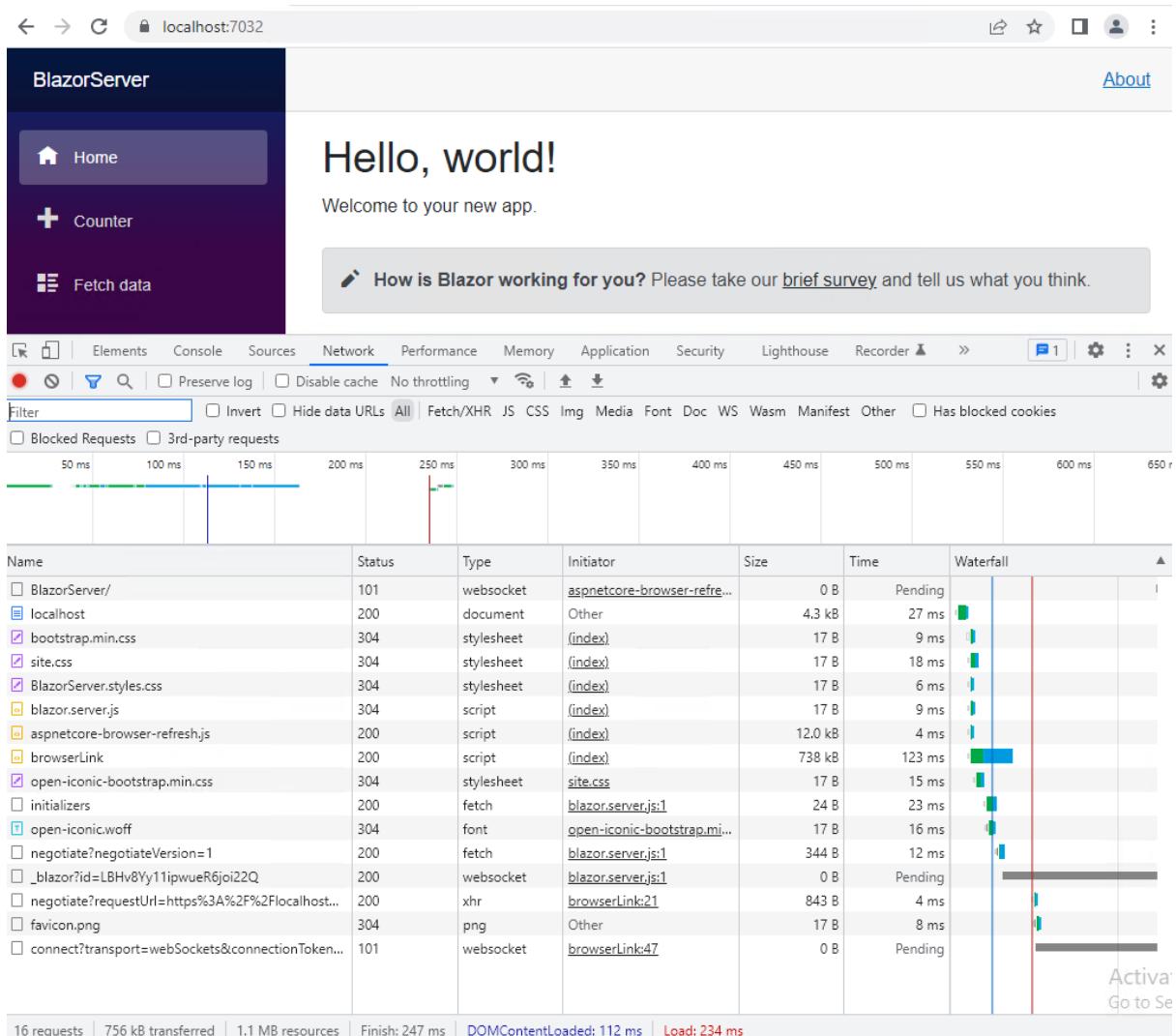
Blazor Server

Es el modelo de Blazor en el cual nuestra app corre en el servidor e interactuamos con ella a través de una conexión de SignalR.

Como el usuario no debe descargar las DLLs del código, la carga es más rápida. El debugging es mucho mejor pues los componentes están corriendo en el servidor.

Sin embargo, la escabilidad es baja ya que hasta los clicks de los botones se procesan en el servidor. Otra desventaja es que no se puede correr fuera de línea (offline).

Creamos una nueva app tipo **Blazor Server App**. Una vez creada, vemos que la estructura de archivos es muy similar a la del proyecto **Client** que teníamos antes. Si lo corremos, veríamos que es muy parecido al proyecto anterior. Si presionamos F12 y vamos a Network, vemos que no se baja un wasm ni librerías. Y que el download fue de sólo 756 kB.



Un código como el siguiente, no funcionaría:

Index.razor

```
@page "/"
@inject IJSRuntime js

<PageTitle>Index</PageTitle>

<h1>Hello, world!</h1>

Welcome to your new app.

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

@code{
    protected async override Task OnInitializedAsync() {
        await js.InvokeVoidAsync("console.log", "hola mundo");
    }
}
```



A screenshot of a browser window showing an error message: "An unhandled exception occurred while processing the request." Below the message, there is a stack trace and a note about JavaScript interop calls being static.

InvalidOperationException: JavaScript interop calls cannot be issued at this time. This is because the component is being statically rendered. When prerendering is enabled, JavaScript interop calls can only be performed during the OnAfterRenderAsync lifecycle method.

Microsoft.AspNetCore.Components.Server.Circuits.RemoteJSRuntime.BeginInvokeJS(long asyncHandle, string identifier, string argsJson, JSCallResultType resultType, long targetInstanceId)

Stack Query Cookies Headers Routing

InvalidOperationException: JavaScript interop calls cannot be issued at this time. This is because the component is being statically rendered. When prerendering is enabled, JavaScript interop calls can only be performed during the OnAfterRenderAsync lifecycle method.

Microsoft.AspNetCore.Components.Server.Circuits.RemoteJSRuntime.BeginInvokeJS(long asyncHandle, string identifier, string argsJson, JSCallResultType resultType, long targetInstanceId)
Microsoft.JSInterop.JSRuntime.InvokeAsync< TValue >(long targetInstanceId, string identifier, CancellationToken cancellationToken, object[] args)
Microsoft.JSInterop.JSRuntime.InvokeAsync< TValue >(long targetInstanceId, string identifier, object[] args)
System.Threading.Tasks.ValueTask< TResult >.get_Result()
Microsoft.JSInterop.JSRuntimeExtensions.InvokeVoidAsync(IJSRuntime jsRuntime, string identifier, object[] args)
BlazorServer.Pages.Index.OnInitializedAsync() in **Index.razor**
+ | 14. await js.InvokeVoidAsync("console.log", "hola mundo");
Microsoft.AspNetCore.Components.ComponentBase.RunInitAndSetParametersAsync()
Microsoft.AspNetCore.Components.Rendering.ComponentState.SetDirectParameters(ParameterView parameters)
Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.InitializeNewComponentFrame(ref DiffContext diffContext, int frameIndex)
Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.InitializeNewSubtree(ref DiffContext diffContext, int frameIndex)
Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.InsertNewFrame(ref DiffContext diffContext, int newFrameIndex)
Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.AppendDiffEntriesForRange(ref DiffContext diffContext, int oldStartIndex, int oldEndIndexExcl, int newstartIndex, int newEndIndexExcl)
Microsoft.AspNetCore.Components.RenderTree.RenderTreeDiffBuilder.ComputeDiff(Renderer renderer, RenderBatchBuilder batchBuilder, int componentId, ArrayRange< RenderTreeFrame > oldTree, ArrayRange< RenderTreeFrame > newTree)
Microsoft.AspNetCore.Components.Rendering.ComponentState.RenderIntoBatch(RenderBatchBuilder batchBuilder, RenderFragment renderFragment, out Exception renderFragmentException)
Microsoft.AspNetCore.Components.RenderTree.Renderer.ProcessRenderQueue()

Esto se debe a que el evento se ejecuta cuando el componente ha sido inicializado **en el servidor**. Todavía no tenemos una conexión con el navegador del usuario durante esta etapa del ciclo de vida del componente. Debemos utilizar **OnAfterRenderAsync**.

Index.razor

```
@page "/"
@inject IJSRuntime js

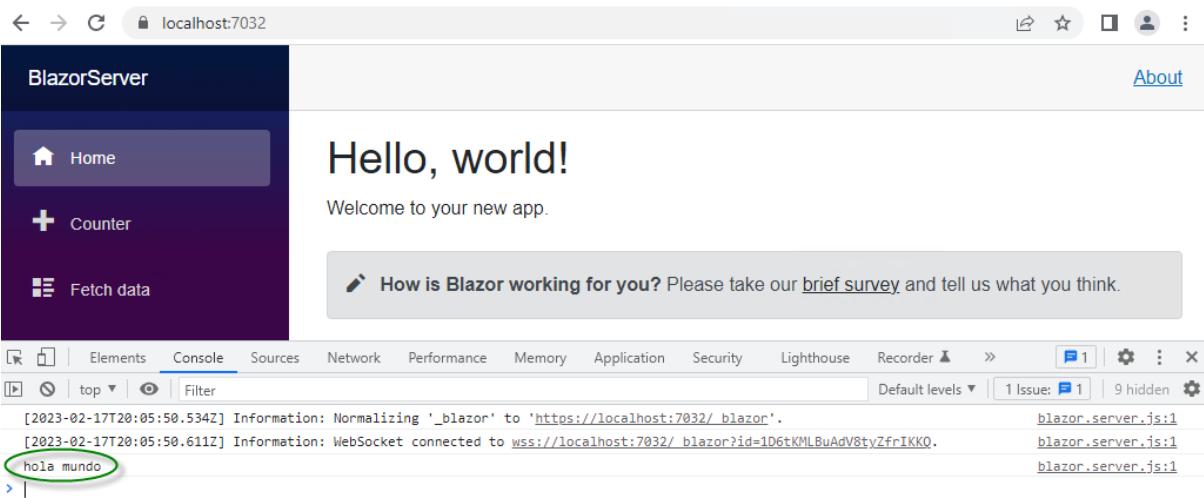
<PageTitle>Index</PageTitle>

<h1>Hello, world!</h1>

Welcome to your new app.

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

```
@code{
    protected async override Task OnAfterRenderAsync(bool firstRender) {
        await js.InvokeVoidAsync("console.log", "hola mundo");
    }
}
```



Las funciones de JS no se pueden ejecutar antes de tener la conexión con el navegador. Eso ocurre después del render del componente. Sin embargo, se puede ejecutar funciones de JS luego de que el usuario cliqueó un botón. Porque si lo hizo... es porque ya terminó de cargarse la interfaz del usuario.

Forzaremos un error para mostrar como se vería:

```
Counter.razor
@page "/counter"

<PageTitle>Counter</PageTitle>

<h1>Counter</h1>

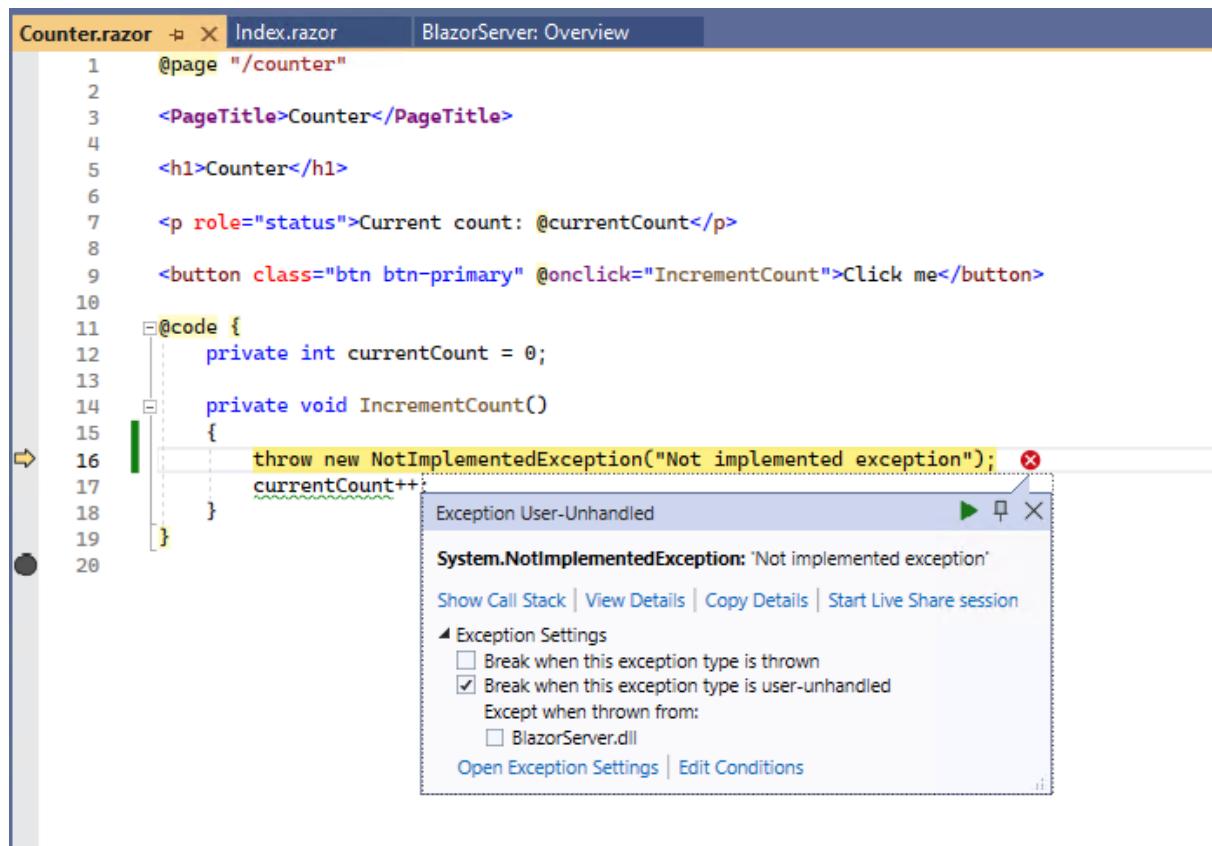
<p role="status">Current count: @currentCount</p>

<button class="btn btn-primary" @onclick="IncrementCount">Click me</button>

@code {
    private int currentCount = 0;

    private void IncrementCount()
    {
        throw new NotImplementedException("Not implemented exception");
        currentCount++;
    }
}
```

Al correr el código y presionar el botón del componente Counter:



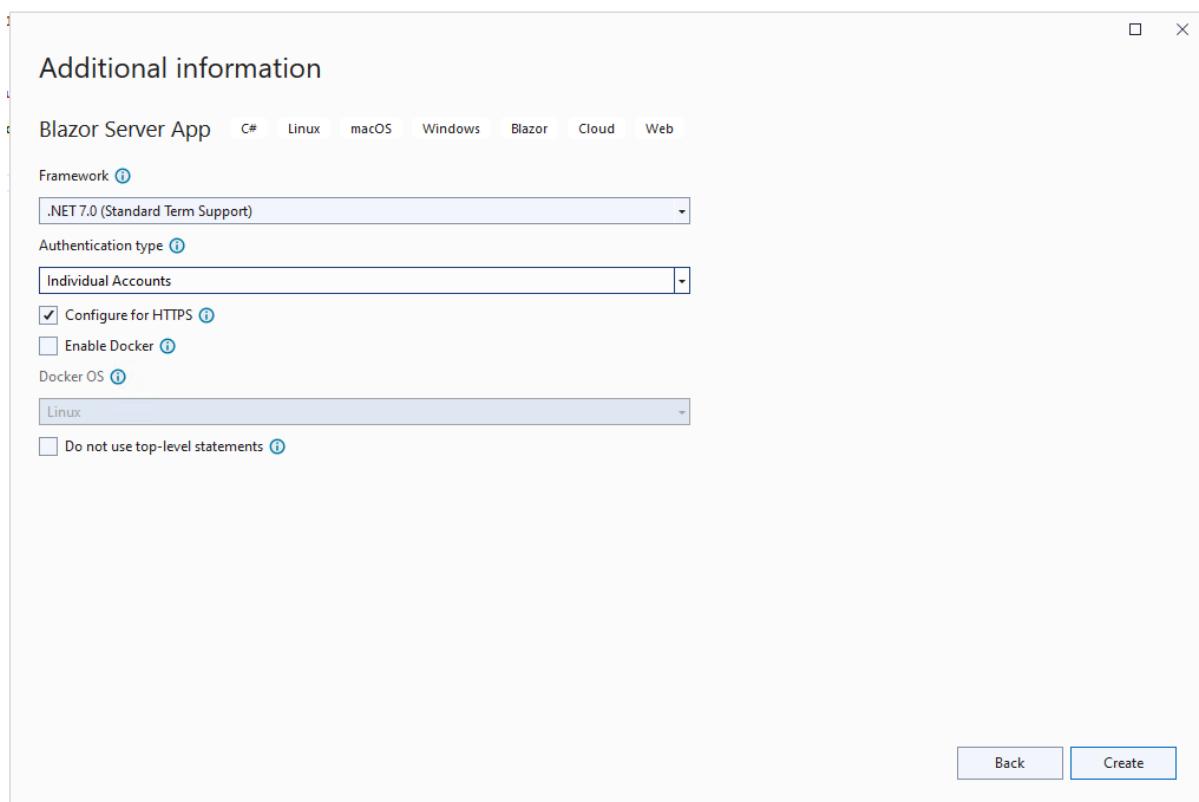
The screenshot shows the Visual Studio code editor with the file 'Counter.razor' open. The code defines a component with an H1 title, a status paragraph, and a button that increments a private variable 'currentCount'. A tooltip is displayed over the line 'throw new NotImplementedException("Not implemented exception");' in the code editor, providing details about the exception and its settings.

```
1 @page "/counter"
2
3 <PageTitle>Counter</PageTitle>
4
5 <h1>Counter</h1>
6
7 <p role="status">Current count: @currentCount</p>
8
9 <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
10
11 @code {
12     private int currentCount = 0;
13
14     private void IncrementCount()
15     {
16         throw new NotImplementedException("Not implemented exception");
17         currentCount++;
18     }
19 }
20
```

Exception User-Unhandled
System.NotImplementedException: 'Not implemented exception'
Show Call Stack | View Details | Copy Details | Start Live Share session
Exception Settings
 Break when this exception type is thrown
 Break when this exception type is user-unhandled
Except when thrown from:
 BlazorServer.dll
Open Exception Settings | Edit Conditions

Nueva app

Crearemos un nuevo proyecto y elegiremos que la autenticación sea **Individual Accounts** porque queremos tener una base de datos con tablas para guardar los usuarios y permitir sus login.

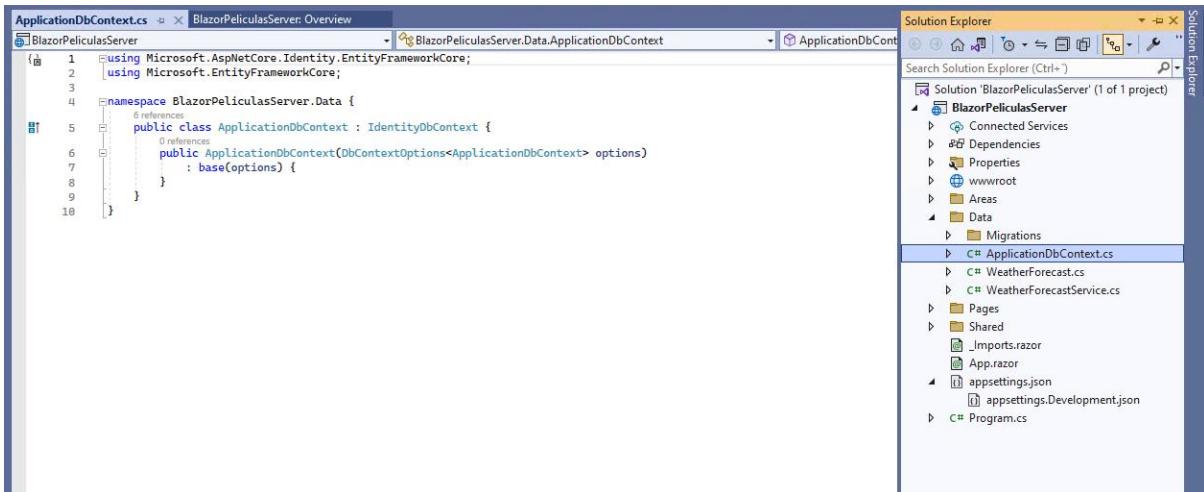


Lo primero que haremos es modificar el string de conexión para que apunte a la BD que teníamos anteriormente:

appsettings.json

```
{  
  "ConnectionStrings": {  
    "DefaultConnection":  
      "Server=(localdb)\\mssqllocaldb;Database=BlazorPeliculas;Trusted_Connection=True;MultipleActiveResultSets=true"  
  },  
  "Logging": {  
    "LogLevel": {  
      "Default": "Information",  
      "Microsoft.AspNetCore": "Warning"  
    }  
  },  
  "AllowedHosts": "*"  
}
```

Si miramos el **Solution Explorer**, vemos que hay una carpeta **Data** y que en ésta está el **ApplicationDbContext** que hereda de **IdentityDbContext** lo que quiere decir que el sistema ya viene configurado el sistema de usuarios:



En **Program.cs** vemos que está el **AddDefaultIdentity**, **AddEntityFrameworkStores** y el middleware de **UseAuthorization**.

Program.cs

```

using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationDbContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= true)
    .AddEntityFrameworkStores<ApplicationDbContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();

var app = builder.Build();

```

```
// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Creamos la carpeta **Entities** en el proyecto y pegamos todos los archivos del proyecto anterior. Luego entramos a cada una de ellas y cambiamos el namespace. Por ej:

Genre.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculasServer.Entities {
    public class Genre {
        public int ID { get; set; }

        [Required(ErrorMessage = "The field '{0}' is required.")]
        public string Name { get; set; } = null!;
        public List<GenresMovie> Genres { get; set; } = new List<GenresMovie>();
    }
}
```

Abrimos el **ApplicationDbContext.cs** y pegamos el código del proyecto anterior. Nos aseguramos que el **namespace** y el **using** a las entidades sean correctos:

ApplicationContext.cs

```
using BlazorPeliculasServer.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculasServer.Data {
    public class ApplicationDbContext : IdentityDbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

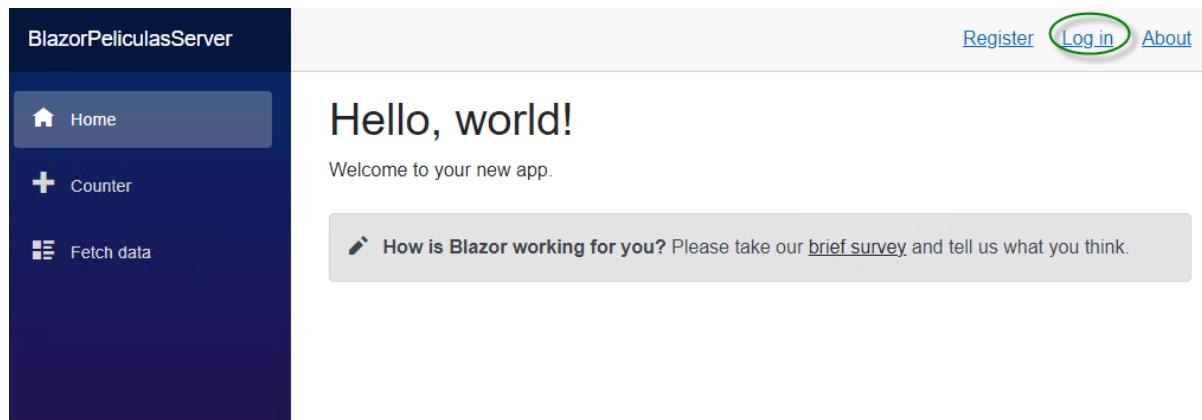
        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder); //No se puede eliminar esta línea.

            modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });
            modelBuilder.Entity<MovieActor>().HasKey(x => new { x.ActorID, x.MovieID });

        }

        public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase Genre.
        public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
        public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase Movie.
        public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla GenresMovie a partir de la clase GenresMovie.
        public DbSet<MovieActor> MoviesActors => Set<MovieActor>(); //Creamos la tabla MoviesActors a partir de la clase MovieActor.
        public DbSet<VoteMovie> VotesMovies => Set<VoteMovie>(); //Creamos la tabla VotesMovies a partir de la clase VoteMovie.
    }
}
```

Si corremos la app vemos que ya está funcionando:



Hacemos click en **Log in** y nos dirige a la página de log in por defecto:

localhost:7210/Identity/Account/Login

BlazorPeliculasServer

Register Login

Log in

Use a local account to log in.

Email
emiliano@blazor.com

Password

Remember me?

Log in

[Forgot your password?](#)

[Register as a new user](#)

[Resend email confirmation](#)

Luego de poner las credenciales, ya estamos logueados:

brief survey and tell us what you think.'"/>

BlazorPeliculasServer

Hello, world!

Welcome to your new app.

How is Blazor working for you? Please take our [brief survey](#) and tell us what you think.

[Home](#)

[Counter](#)

[Fetch data](#)

Hello, emiliano@blazor.com! Log out About

Abrimos el código de NavMenu.razor y pegamos en él, el código del proyecto anterior. Como tendremos que hacer cambios a la forma de obtener los roles, por el momento quitamos el **roles="admin"** del componente **AuthorizeView**:

Logout.razor

```
<div class="top-row ps-3 navbar navbar-dark">
<div class="container-fluid">
<a class="navbar-brand" href="">BlazorPeliculas</a>
<button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
    <span class="navbar-toggler-icon"></span>
</button>
```

```
</div>
</div>

<div class="@NavControllerCssClass nav-scrollable" @onclick="ToggleNavController">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="oi oi-plus" aria-hidden="true"></span> Counter
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="fetchdata">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="movies/filter">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Movies filter
            </NavLink>
        </div>

        <AuthorizeView>
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="genres">
                    <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
                </NavLink>
            </div>
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="actors">
                    <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
                </NavLink>
            </div>
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="movies/create">
                    <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
                </NavLink>
            </div>
            <div class="nav-item px-3">
                <NavLink class="nav-link" href="users">
                    <span class="oi oi-list-rich" aria-hidden="true"></span> Users
                </NavLink>
            </div>
        </AuthorizeView>
```

```
</nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu() {
        collapseNavMenu = !collapseNavMenu;
    }
}
```

Migrando usuarios

Copiamos la carpeta **Client\Pages\Users** del proyecto anterior en la carpeta **Pages** del proyecto actual. Si abriéramos el objeto **UsersList.razor** veríamos que hay muchos errores. Por ej: no tenemos el componente de paginación.

Vamos a **Client\Shared** del proyecto anterior y copiamos todos los componentes. Los pegamos en **Shared** del proyecto actual sin sobre-escribir **MainLayout.razor**, **NavBar.razor**, **MainLayout.razor.css**, **NavBar.razor.css** o **SurveyPrompt.razor**.

Vamos a **Client\Helpers** del proyecto anterior y copiamos todos los componentes. Los pegamos en el proyecto actual. Abrimos los archivos y corregimos los **namespaces**. Agregamos el **namespace** de **Helpers** en **_Imports.razor**.

```
_Imports.razor
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.JSInterop
@using BlazorPeliculasServer
@using BlazorPeliculasServer.Shared
@using BlazorPeliculasServer.Helpers
```

Para los mensajes estamos usando el SweetAlert. En este proyecto no hay un index.html pero sí un **_Host.cshtml**.

Buscamos el paquete razor/sweetalert2 (de CurrieTechnologies) y lo instalamos. Luego hacemos unos cambios para utilizarlo.

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");
builder.Services.AddDbContext<ApplicationContext>(options =>
    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Vamos a <https://github.com/Basaingeal/Razor.SweetAlert2> para copiar el código de Add the script tag.

```
_Host.cshtml
@page "/"
@using Microsoft.AspNetCore.Components.Web
@namespace BlazorPeliculasServer.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <base href="/" />
    <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
    <link href="css/site.css" rel="stylesheet" />
    <link href="BlazorPeliculasServer.styles.css" rel="stylesheet" />
    <link rel="icon" type="image/png" href="favicon.png"/>
    <component type="typeof(HeadOutlet)" render-mode="ServerPrerendered" />
</head>
<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">X </a>
    </div>

    <script src="_framework/blazor.server.js"></script>
    <script src="_content/CurrieTechnologies.Razor.SweetAlert2/sweetalert2.min.js"></script>
</body>
</html>
```

Y el **using** en el `_Imports.razor`.

```
_Imports.razor
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.JSInterop
@using BlazorPeliculasServer
@using BlazorPeliculasServer.Shared
@using BlazorPeliculasServer.Helpers
@using CurrieTechnologies.Razor.SweetAlert2
```

Para la selección de actores utilizaremos un componente de selección múltiple. **Typeahead**, por ejemplo, es básicamente cuando tú tienes un textbox, tú escribes en él y según lo que tú escribas te salen sugerencias. Vamos a <https://github.com/Blazored/Typeahead> para instalarlo. En ella dice que se puede instalar bajando el paquete NuGet **Blazored.Typeahead**. En esa página dice que se deben agregar las siguientes líneas en el `index.html`.

```
_Host.cshtml
@page "/"
@using Microsoft.AspNetCore.Components.Web
@namespace BlazorPeliculasServer.Pages
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <base href="/" />
    <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />
    <link href="css/site.css" rel="stylesheet" />
    <link href="BlazorPeliculasServer.styles.css" rel="stylesheet" />
    <link rel="icon" type="image/png" href="favicon.png"/>
    <link href="_content/Blazored.Typeahead/blazored-typeahead.css" rel="stylesheet" />
    <component type="typeof(HeadOutlet)" render-mode="ServerPrerendered" />
</head>
<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">X </a>
</div>

<script src="_framework/blazor.server.js"></script>
<script src="_content/CurrieTechnologies.Razor.SweetAlert2/sweetalert2.min.js"></script>
<script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>
</body>
</html>
```

Para poder mostrar el preview crearemos el componente **ShowMD** en Shared e instalamos el NuGet package llamado **Markdig** (Markdig Mutel).

Hay una diferencia fundamental entre el proyecto anterior y este. No es necesario hacer peticiones HTTP para traer listados o agregar registros **porque ya estamos en el servidor**.

Creamos la carpeta **Repositories** y agregamos la clase **UsersRepository.cs**. Agregamos el using en **_Imports.razor**.

```
_Imports.razor
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.JSInterop
@using BlazorPeliculasServer
@using BlazorPeliculasServer.Shared
@using BlazorPeliculasServer.Helpers
@using CurrieTechnologies.Razor.SweetAlert2
@using BlazorPeliculasServer.Repositories
```

Agregaremos el servicio de repositorio de usuarios como transitorio (Transient). Esto es que cuando utilizamos **Scoped** es porque queremos encapsular el tiempo de vida de un objeto servido a través de nuestro sistema de inyección de dependencias en un contexto específico. En el caso de hace **ASP.NET Core**, ese contexto es el tiempo de vida de la petición http que le corresponde al usuario.

Sin embargo, en **Blazor-Serverside** no tenemos ese contexto **http** porque la aplicación ya se encuentra en el servidor y no utiliza **http** para comunicarse el componente con el repositorio. Sino que es simplemente una invocación de una clase.

Por tanto, para no tener un contexto de datos permanentemente vivo en memoria (lo cual nos puede causar errores) lo que hacemos es utilizar transitorio para que cada vez que queramos utilizar el contexto de datos a través de la inyección de dependencias, obtendremos una nueva instancia.

Esto va a hacer que no tengamos que cambiar nada de nuestro código que teníamos en nuestra aplicación de hace **ASP.NET Core** en nuestro proyecto de **Blazor- WebAssembly**.

Esto lo hacemos para evitar errores, ya que en **Entity Framework Core** pueden ocurrir errores si tú cargas dos veces una misma entidad en memoria. Esto no nos ocurría antes porque siempre estábamos dentro del contexto de una petición **http**.

Pero este no es el caso de **Blazor-Serverside**. Así que por eso la recomendación es en **Blazor-Serverside** utilizar **Transient**.

También tenemos que colocar nuestro **ApplicationDbContext** como **Transient**. Por defecto **AddDbContext** se coloca como **Scoped** pero lo queremos como **Transient**.

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationDbContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationDbContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationDbContext>(options =>
//    options.UseSqlServer(connectionString));
```

```
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Copiamos el contenido de la **UserController.cs** que teníamos antes y lo pegamos en la **UsersRepository.cs**. Por empezar, el constructor debe ser **UsersRepository**.

No necesitamos **ActionResult** para nada ni **FromQuery** (son conceptos de MVC pero esto es una clase normal).

Quitamos los atributos de **HttpGet**, **HttpPost**, etc.

Agregamos la carpeta **DTO** y modificamos los namespaces. Agregamos el **using** en **_Imports.razor**. Lo mismo para el namespace de entidades.

_Imports.razor

```
@using System.Net.Http  
@using Microsoft.AspNetCore.Authorization  
@using Microsoft.AspNetCore.Components.Authorization  
@using Microsoft.AspNetCore.Components.Forms  
@using Microsoft.AspNetCore.Components.Routing  
@using Microsoft.AspNetCore.Components.Web  
@using Microsoft.AspNetCore.Components.Web.Virtualization  
@using Microsoft.JSInterop  
@using BlazorPeliculasServer  
@using BlazorPeliculasServer.Shared  
@using BlazorPeliculasServer.Helpers  
@using CurrieTechnologies.Razor.SweetAlert2  
@using BlazorPeliculasServer.Repositories  
@using BlazorPeliculasServer.DTOs  
@using BlazorPeliculasServer.Entities
```

Agregamos el using de BlazorPeliculasServer.DTOs y el de EFC. Lo primero que tenemos que ver es que ya no necesitamos **InserPaginationParametersInResponse** de una respuesta **HTTP** porque ya no hay una respuesta **http**. Recordamos que dijimos que nuestros componentes se van a comunicar directamente con nuestros repositorios.

Por tanto, eso no lo necesitamos como tal, sino que solamente necesitamos el cálculo del total de páginas obtenido a partir del queryable.

Por tanto, lo que vamos a hacer es que vamos a crear una nueva clase a en la carpeta **Helpers** que nos va a ayudar con eso. La llamaremos **IQueryableExtensions.cs** y será pública y estática. Crearemos, en ella, la función para calcular la cantidad de páginas necesarias y copiamos el método **ToPage** del proyecto anterior.

IQueryableExtensions.cs

```
using BlazorPeliculasServer.DTOs;  
using Microsoft.EntityFrameworkCore;  
  
namespace BlazorPeliculasServer.Helpers {  
    public static class IQueryableExtensions {  
        public async static Task<int> CalculateTotalPages<T> (this IQueryable<T> queryable, int  
recordsToShowCount) {  
            double account = await queryable.CountAsync();  
            int totalPages = (int)Math.Ceiling(account / recordsToShowCount);  
            return totalPages;  
        }  
  
        public static IQueryable<T> ToPage<T>(this IQueryable<T> queryable, PaginationDTO pagination)  
        {  
            return queryable
```

```
.Skip((pagination.Page - 1) * pagination.RecordCount)
    .Take(pagination.RecordCount);
}
}
```

Ahora debemos pensar cómo vamos a hacer para retornar tanto el listado de registros que el usuario puede ver en ese momento y el total de páginas. Para eso vamos a crear un nuevo tipo que se llama Respuesta Patinada.

Para eso vamos a crear un nuevo tipo llamada **PaginatedResponse.cs** en la carpeta **DTOs**. Es de `<T>` porque servirá para paginar todo tipo de elemento que queramos:

PaginatedResponse.cs

```
namespace BlazorPeliculasServer.DTOs {
    public class PaginatedResponse<T> {
        public int totalPages { get; set; }
        public List<T> Records { get; set; }
    }
}
```

El Get de usuario devolverá un elemento tipo **PaginatedResponse<UserDTO>**. Eliminamos los return `NoContent()` y agregaremos un nuevo Claim específico al momento de devolver los roles.

UsersRepository.razor

```
namespace BlazorPeliculasServer.Repositories {
    public class UsersRepository {
        private readonly ApplicationDbContext context;
        private readonly UserManager<IdentityUser> userManager;

        public UsersRepository(ApplicationDbContext context,
            UserManager<IdentityUser> userManager) {
            this.context = context;
            this.userManager = userManager;
        }

        public async Task<PaginatedResponseDTO<UserDTO>> Get(PaginationDTO pagination) {
            var queryable = context.Users.AsQueryable();
            var response = new PaginatedResponseDTO<UserDTO>();

            response.totalPages = await queryable.CalculateTotalPages(pagination.RecordCount);
            response.Records = await queryable.ToPage(pagination)
                .Select(x => new UserDTO { ID = x.Id, Email = x.Email! })
                .ToListAsync();
            return response;
        }
    }
}
```

```
public async Task<List<RoleDTO>> GetRoles() {
    return await context.Roles.Select(x => new RoleDTO { Name = x.Name! }).ToListAsync();
}

public async Task AssignRoleToUser(EditRoleDTO editRoleDTO) {
    var user = await userManager.FindByIdAsync(editRoleDTO.UserID);
    if(user is null)
        throw new Exception("User was not found!");

    await userManager.AddClaimAsync(user, new Claim(ClaimTypes.Role, editRoleDTO.Role));

    await userManager.AddToRoleAsync(user, editRoleDTO.Role);
}

public async Task RemoveRoleFromUser(EditRoleDTO editRoleDTO) {
    var user = await userManager.FindByIdAsync(editRoleDTO.UserID);
    if(user is null)
        throw new Exception("User was not found!");

    await userManager.RemoveClaimAsync(user, new Claim(ClaimTypes.Role, editRoleDTO.Role));
    await userManager.RemoveFromRoleAsync(user, editRoleDTO.Role);
}
```

En **UsersList.cs** vamos a ver que ya no vamos a utilizar eso de Get pasandole un URL, sino que sencillamente vamos a invocar el método de paginación de nuestro repositorio de usuarios.

UsersList.cs

```
@page "/users"
@inject UsersRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swal

<h3>Users List</h3>

<Pagination ActualPage="currentPage" TotalPages="totalPages"
SelectedPage="selectedPage"></Pagination>

<GenericList List="Users">
<HasRecordsComplete>
<table class="table">
<thead>
<tr>
<th></th>
<th>User</th>
</tr>
```

```

</thead>
<tbody>
    @foreach(var user in Users!) {
        <tr>
            <td>
                <a href="/users/edit/@user.ID" class="btn btn-success">Edit</a>
            </td>
            <td>
                @user.Email
            </td>
        </tr>
    }
</tbody>
</table>
</HasRecordsComplete>
</GenericList>

@code {
    List<UserDTO>? Users;
    private int currentPage = 1;
    private int totalPages;

    protected override async Task OnInitializedAsync() {
        await selectedPage(1);
    }

    private async Task selectedPage(int page) {
        currentPage = page;
        await Load(page);
    }

    private async Task Load(int page = 1) {
        var paginatedResponse = await repository.Get(new DTOs.PaginationDTO { Page = page });

        totalPages = paginatedResponse.totalPages;
        Users = paginatedResponse.Records;
    }
}

```

Modificamos el componente **EditUser.razor**. Movemos el código de validación de edición de rol a un método llamado **ValidateEditRole** y eliminamos el método **EditRole**.

EditUser.razor

```

@page "/users/edit/{UserID}"
@inject UserRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swal

```

```
<h3>Edit User</h3>

@if(roles is null) {
    <LoadingWheel/>
}
else {
    <div class="form-inline">
        <select class="form-select mb-2" @bind="selectedRole">
            <option value="0">-- Select a role--</option>
            @foreach(var rol in roles) {
                <option value="@rol.Name">@rol.Name</option>
            }
        </select>

        <button class="btn btn-info mb-2" @onclick="AssignRole">Assign role</button>
        <button class="btn btn-danger mb-2" @onclick="RemoveRole">Remove role</button>
    </div>
}

@code {
    [Parameter]
    public string UserID { get; set; } = null!;
    private List<RoleDTO>? roles;
    private string selectedRole = "0";

    protected override async Task OnInitializedAsync() {
        roles = await repository.GetRoles();
    }

    private async Task<bool> ValidateEditRole() {
        if(selectedRole == "0") {
            await swal.FireAsync("Error", "You must select a role", SweetAlertIcon.Error);
            return false;
        }

        return true;
    }

    private async Task AssignRole() {
        if(!await ValidateEditRole())
            return;

        var roleDTO = new EditRoleDTO() { Role = selectedRole, UserID = UserID };
        await repository.AssignRoleToUser(roleDTO);
        await swal.FireAsync("Success", "Role was edited successfully", SweetAlertIcon.Success);
    }

    private async Task RemoveRole() {
        if(!await ValidateEditRole())
            return;
```

```
return;

var roleDTO = new EditRoleDTO() { Role = selectedRole, UserID = UserID };
await repository.RemoveRoleFromUser(roleDTO);
await swal.fireAsync("Success", "Role was edited successfully", SweetAlertIcon.Success);
}

}
```

Finalmente, agregamos el rol IdentityRole en la clase [Program.cs](#).

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.Sweetalert2;
using BlazorPeliculasServer.Repositories;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationContext>(options =>
//    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();

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

```
// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Corremos el proyecto, nos loguemos y vamos a la parte de usuarios:

User
emiliano@blazor.com

Migrando géneros

Copiamos la carpeta **Client\Pages\Genres** del proyecto anterior en la carpeta **Pages** del proyecto actual. Lo primero que necesitamos es un repositorio para géneros. Creamos la clase **GenresRepository.cs** en la carpeta **Repositories**. Copiamos en ella el código de **GenresControllers.cs** del proyecto anterior.

Eliminamos los atributos **HttpGet**, **HttpPost**, **AllowAnonymous**, etc. Traemos los namespaces de **Microsoft.AspNetCore.Mvc** y **Microsoft.EntityFrameworkCore**. Eliminamos los **ActionResult**.

Cambiamos los return **NotFound** por Exceptions.

GenresRepository.cs

```
using BlazorPeliculasServer.Data;
using BlazorPeliculasServer.Entities;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculasServer.Repositories {
    public class GenresRepository {
        private readonly ApplicationDbContext context;
        public GenresRepository(ApplicationDbContext context) {
            this.context = context;
        }

        public async Task<List<Genre>> Get() {
            return await context.Genres.ToListAsync();
        }

        public async Task<Genre> Get(int id) {
            var genre = await context.Genres.FirstOrDefaultAsync(genre => genre.ID == id);

            if(genre is null)
                throw new ApplicationException($" Genre {id} was not found");

            return genre;
        }

        [HttpPost]
        public async Task<ActionResult<int>> Post(Genre genre) {
            context.Add(genre);
            await context.SaveChangesAsync();
            return genre.ID;
        }

        public async Task Put(Genre genre) {
            context.Update(genre); //Marco el género para ser actualizado.
            await context.SaveChangesAsync(); //Se hace el UPDATE
        }
    }
}
```

```
}
```

```
public async Task Delete(int id) {
    var affectedFiles = await context.Genres
        .Where(x => x.ID == id)
        .ExecuteDeleteAsync();

    if(affectedFiles == 0)
        throw new ApplicationException($" Genre {id} was not found");
}
```

```
}
```

GenresList.razor

```
@page "/genres"
@using Microsoft.AspNetCore.Authorization;
@inject GenresRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]

<h3>Genres</h3>

<div class="mb-3">
    <a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

<GenericList List="Genres">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach(var item in Genres!) {
                    <tr>
                        <td>
                            <a href="/genres/edit/@item.ID" class="btn btn-success">Edit</a>
                            <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                        </td>
                        <td>@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>
```

```
@code {
    public List<Genre>? Genres { get; set; }

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task Load() {
        Genres = await repository.Get();
    }

    public async Task Delete(Genre genre) {
        await repository.Delete(genre.ID);
        await Load();
    }
}
```

CreateGenre.razor

```
@page "/genres/create"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject GenresRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]

<h3>Create Genre</h3>

<GenreForm @ref="genreForm" Genre="genre" OnValidSubmit="Create" />

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío
    private GenreForm? genreForm;

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private async Task Create() {
        try {
            await repository.Post(genre);

            genreForm!.formPostedCorrectly = true;
            navManager.NavigateTo("/genres");
        }
        catch(Exception ex) {
            await swAl.FireAsync("Error", ex.Message.ToString(), SweetAlertIcon.Error);
        }
    }
}
```

EditGenre.razor

```
@page "/genres/edit/{GenreID:int}"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject GenresRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]

<h3>Edit genre</h3>
@if(Genre is not null) {
    <GenreForm @ref="genreForm" Genre="Genre" OnValidSubmit="Edit" />
}
else {
    <LoadingWheel/>
}

@code {
    [Parameter] public int GenreID { get; set; }
    private Genre? Genre;
    private GenreForm? genreForm;

    protected override async Task OnInitializedAsync() {
        Genre = await repository.Get(GenreID);

        if(Genre is null)
            navManager.NavigateTo("genres");
    }

    private async Task Edit() {
        await repository.Put(Genre);
        genreForm!.formPostedCorrectly = true;
        navManager.NavigateTo("/genres");
    }
}
```

Agregamos el repositorio como servicio:

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;

var builder = WebApplication.CreateBuilder(args);
```

```
// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationContext>(options =>
//    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();
builder.Services.AddTransient<GenresRepository>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

AsNoTracking

En el código, necesita agregar **AsNoTracking()** para que el método de Delete funcione correctamente. Esto lo hace tanto en ambos Gets. Según explica es para que no le dé seguimiento. Con el código tal cuál como estamos (posiblemente en una versión más nueva que la del docente), no fue necesario. Tenemos que tener cuidado pues en ocasiones quizás por error el tracker Framework Core le va a estar dando seguimiento a una entidad a la cual no quisiéramos que le esté dando seguimiento porque eso puede hacer que tengamos errores en tiempo de ejecución.

Migrando actores

Copiamos la carpeta **Client\Pages\Actors** del proyecto anterior en la carpeta **Pages** del proyecto actual. Lo primero que necesitamos es un repositorio para actores. Creamos la clase **ActorsRepository.cs** en la carpeta **Repositories**. Copiamos en ella el código de **ActorsControllers.cs** del proyecto anterior.

Copiamos los archivos **IFileSaver.cs**, **FileServerLocal.cs** y **FileSaveAzStorage.cs** desde **Client\Helpers** a **Helpers**. Lo mismo con **AutoMapperProfile.cs**. Modificamos los namespaces de los primeros 3 y los using del cuarto.

Agregamos el NuGet Package llamado **Azure.Storage.Blobs**. Instalamos el NuGet: **AutoMapper.Extensions.Microsoft.DependencyInjection**

Agregamos el servicio de **FileSaver** y el AutoMapper en **Program.cs**.

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;
using BlazorPeliculasServer.Helpers;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationDbContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationDbContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationDbContext>(options =>
```

```
// options.UseSqlServer(connectionString);
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();
builder.Services.AddTransient<GenresRepository>();
builder.Services.AddScoped<IFileSaver, FileSaverLocal>();
builder.Services.AddAutoMapper(typeof(Program));

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Eliminamos los atributos **HttpGet**, **HttpPost**, **AllowAnonymous**, etc. Traemos los namespaces de **Microsoft.AspNetCore.Mvc** y **Microsoft.EntityFrameworkCore**. Eliminamos los **ActionResult**, **FromQuery** y **NoContent()**.

ActorsRepository.cs

```
using AutoMapper;
using BlazorPeliculasServer.Data;
using BlazorPeliculasServer.DTOs;
using BlazorPeliculasServer.Entities;
using BlazorPeliculasServer.Helpers;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculasServer.Repositories {
    public class ActorsRepository {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly string container = "people";

        public ActorsRepository(ApplicationDbContext context, IFileSaver fileSaver, IMapper mapper) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
        }

        public async Task<PaginatedResponseDTO<Actor>> Get(PaginationDTO pagination) {
            //return await context.Actors.ToListAsync();
            var queryable = context.Actors.AsQueryable();
            var paginatedResponse = new PaginatedResponseDTO<Actor>();
            paginatedResponse.totalPages = await queryable.CalculateTotalPages(pagination.RecordCount);
            paginatedResponse.Records = await queryable.OrderBy(x =>
                x.Name).ToPage(pagination).ToListAsync();

            return paginatedResponse;
        }

        public async Task<List<Actor>> Get(string searchText) {
            if(string.IsNullOrWhiteSpace(searchText))
                return new List<Actor>();

            searchText = searchText.ToLower();
            return await context.Actors
                .Where(x => x.Name.ToLower().Contains(searchText))
                .Take(5)
                .ToListAsync();
        }

        public async Task<Actor> Get(int id) {
            return await context.Actors.FirstOrDefaultAsync(actor => actor.ID == id); //Si no lo encuentra,
devuelve NULL
        }

        public async Task<int> Post(Actor actor) {
```

```
if(!string.IsNullOrWhiteSpace(actor.Photo)) {
    //Nos mandaron una foto desde el frontend
    var photoActor = Convert.FromBase64String(actor.Photo);
    actor.Photo = await fileSaver.SaveFile(photoActor, "jpg", container);
}

context.Add(actor);
await context.SaveChangesAsync();
return actor.ID;
}

public async Task Put(Actor actor) {
    var actorDB = await context.Actors.FirstOrDefaultAsync(a => a.ID == actor.ID);

    if(actorDB is null)
        throw new ApplicationException($"Actor {actor.ID} was not found!");

    //Tomá las propiedades de actor y pasalas a actorDB
    actorDB = mapper.Map(actor, actorDB);

    if(!string.IsNullOrWhiteSpace(actor.Photo)) {
        //Nos mandaron una foto desde el frontend
        var photoActor = Convert.FromBase64String(actor.Photo);
        actorDB.Photo = await fileSaver.EditFile(photoActor, "jpg", container, actorDB.Photo!);
    }

    await context.SaveChangesAsync(); //Se hace el UPDATE
}

public async Task Delete(int id) {
    var actor = await context.Actors.FirstOrDefaultAsync(x => x.ID == id);

    if(actor is null)
        throw new ApplicationException($"Actor {id} was not found!");

    context.Remove(actor); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if(!string.IsNullOrWhiteSpace(actor.Photo))
        await fileSaver.DeleteFile(actor.Photo!, container);
}
}
```

Agregamos el servicio en [Program.cs](#).

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
```

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;
using BlazorPeliculasServer.Helpers;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationContext>(options =>
//    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();
builder.Services.AddTransient<GenresRepository>();
builder.Services.AddTransient<ActorsRepository>();
builder.Services.AddScoped<IFileSaver, FileSaverLocal>();
builder.Services.AddAutoMapper(typeof(Program));
var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

ActorsList.razor

```
@page "/actors"
@using Microsoft.AspNetCore.Authorization;
@inject ActorsRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swal
@attribute [Authorize(Roles = "admin")]
<h3>Actors list</h3>

<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>

<Pagination ActualPage="ActualPage"
    TotalPages="TotalPages"
    SelectedPage="SelectedPage" />

<GenericList List="Actors">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach (var item in Actors!) {
                    <tr>
                        <td>
                            <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                            <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                        </td>
                        <td>&ampnbsp@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>
```

```
</tr>
}
</tbody>
</table>
</HasRecordsComplete>
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }
    private int ActualPage = 1;
    private int TotalPages;

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task SelectedPage(int page) {
        ActualPage = page;
        await Load(page);
    }

    private async Task Load(int page = 1) {
        var paginatedResponse = await repository.Get(new DTOs.PaginationDTO { Page = page });
        Actors = paginatedResponse.Records;
        TotalPages = paginatedResponse.totalPages;
    }

    public async Task Delete(Actor actor) {
        await repository.Delete(actor.ID);
        await Load();
    }
}
```

CreateActor.razor

```
@page "/actors/create"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject ActorsRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
<h3>Create Actor</h3>
<ActorsForm OnValidSubmit="Create" Actor="Actor" />
@code {
    private Actor Actor = new Actor();

    async Task Create() {
        await repository.Post(Actor);
        navManager.NavigateTo("/actors");
    }
}
```

EditActor.razor

```
@page "/actors/edit/{ActorID:int}"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject ActorsRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
<h3>Edit actor</h3>

@if(Actor is not null) {
    <ActorsForm Actor="Actor" OnValidSubmit="Edit" />
}
else {
    <LoadingWheel/>
}

@code {
    [Parameter] public int ActorID { get; set; }
    Actor? Actor;

    protected override async Task OnInitializedAsync() {
        Actor = await repository.Get(ActorID);

        if(Actor == null)
            navManager.NavigateTo("actors");
    }

    private async Task Edit() {
        await repository.Put(Actor);
        navManager.NavigateTo("/actors");
    }
}
```

Copiamos las carpetas **Server/wwwroot/people** y **Server/wwwroot/movies** del proyecto anterior a la carpeta **wwwroot**.

Creamos el archivo **wwwroot/css/custom.css**.

```
custom.css
.form-markdown {
    display: flex;
}

.form-markdown textarea {
    width: 500px;
    height: 500px;
    margin-right: 15px;
}
```

```
.form-markdown .markdown-container {  
    border: 1px dashed black;  
    width: 500px;  
    height: 500px;  
}
```

Agregamos el stylesheet y el font-awesome:

```
Host.cshtml  
@page "/"  
@using Microsoft.AspNetCore.Components.Web  
@namespace BlazorPeliculasServer.Pages  
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers  
  
<!DOCTYPE html>  
<html lang="en">  
<head>  
    <meta charset="utf-8" />  
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />  
    <base href="/" />  
    <link rel="stylesheet" href="css/bootstrap/bootstrap.min.css" />  
    <link href="css/site.css" rel="stylesheet" />  
    <link href="BlazorPeliculasServer.styles.css" rel="stylesheet" />  
    <link rel="icon" type="image/png" href="favicon.png"/>  
    <link href="_content/Blazored.Typeahead/blazored-typeahead.css" rel="stylesheet" />  
    <link href="css/custom.css" rel="stylesheet" />  
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css" rel="stylesheet"  
/>  
    <component type="typeof(HeadOutlet)" render-mode="ServerPrerendered" />  
</head>  
<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">X </a>  
    </div>  
  
<script src="_framework/blazor.server.js"></script>  
<script src="_content/CurrieTechnologies.Razor.Sweetalert/sweetalert2.min.js"></script>  
<script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>  
</body>  
</html>
```

Migrando películas

Copiamos la carpeta **Client\Pages\Movies** del proyecto anterior en la carpeta **Pages** del proyecto actual. Lo primero que necesitamos es un repositorio para actores. Creamos la clase **MoviesRepository.cs** en la carpeta **Repositories**. Copiamos en ella el código de **MoviesControllers.cs** del proyecto anterior.

Eliminamos los atributos **HttpGet**, **HttpPost**, **AllowAnonymous**, etc. Traemos los namespaces de **Microsoft.AspNetCore.Mvc** y **Microsoft.EntityFrameworkCore**. Eliminamos los **ActionResult**, **FromQuery** y **NoContent()**.

Crearemos un servicio que utilice el **AuthenticationStateProvider** para poder obtener el ID del usuario. Para ello, creamos el archivo **AuthenticationStateService.cs** en la carpeta **Helpers**.

AuthenticationStateService.cs

```
using Microsoft.AspNetCore.Components.Authorization;
using System.Security.Claims;

namespace BlazorPeliculasServer.Helpers {
    public class AuthenticationStateService {
        private readonly AuthenticationStateProvider authStateProvider;

        public AuthenticationStateService(AuthenticationStateProvider authStateProvider) {
            this.authStateProvider = authStateProvider;
        }

        public async Task<string?> GetCurrentUserID() {
            var userState = await authStateProvider.GetAuthenticationStateAsync();
            if(!userState.User.Identity!.IsAuthenticated)
                return null;

            var claims = userState.User.Claims;
            var claimWithUserId = claims.FirstOrDefault(x => x.Type == ClaimTypes.NameIdentifier);

            if(claimWithUserId == null)
                throw new ApplicationException("Could not find User's ID");

            return claimWithUserId.Value;
        }
    }
}
```

Agregamos el servicio en **Program.cs**.

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
```

```
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;
using BlazorPeliculasServer.Helpers;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationContext>(options =>
//    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();
builder.Services.AddTransient<GenresRepository>();
builder.Services.AddTransient<ActorsRepository>();
builder.Services.AddScoped<IFileSaver, FileSaverLocal>();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddTransient<AuthenticationStateService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

MoviesRepository.cs

```
using AutoMapper;
using BlazorPeliculasServer.Data;
using BlazorPeliculasServer.DTOs;
using BlazorPeliculasServer.Entities;
using BlazorPeliculasServer.Helpers;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Infrastructure;

namespace BlazorPeliculasServer.Repositories {
    public class MoviesRepository {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly UserManager<IdentityUser> userManager;
        private readonly AuthenticationStateService authStateService;
        private readonly string container = "movies";

        public MoviesRepository(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper,
            UserManager<IdentityUser> userManager,
            AuthenticationStateService authStateService) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
            this.userManager = userManager;
        }
    }
}
```

```
this.authStateService = authStateService;
}

public async Task<HomePageDTO> Get() {
    var limit = 6;
    var onBoardMovies = await context.Movies
        .Where(movie => movie.OnBillboard)
        .Take(limit)
        .OrderByDescending(movie => movie.ReleaseDate)
        .ToListAsync();
    var today = DateTime.Today;
    var nextReleases = await context.Movies
        .Where(movie => movie.ReleaseDate > today)
        .Take(limit)
        .OrderBy(movie => movie.ReleaseDate)
        .ToListAsync();

    var result = new HomePageDTO {
        OnBoard = onBoardMovies,
        NextReleases = nextReleases
    };
    return result;
}

public async Task<MovieViewDTO> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if(movie is null) {
        //No se encontró la película
        return null;
    }

    var votesMedia = 0.0;
    var userVote = 0;

    if(await context.VotesMovies.AnyAsync(x => x.MovieID == id)) {
        //Alguien ha votado por la película
        votesMedia = await context.VotesMovies
            .Where(x => x.MovieID == id)
            .AverageAsync(x => x.Voto);

        var userID = await authStateService.GetCurrentUserID();

        if(userID != null) {
```

```
var userVoteDB = await context.VotesMovies
    .FirstOrDefaultAsync(x => x.MovieID == id && x.UserID == userID);

    if(userVoteDB is not null)
        userVote = userVoteDB.Voto;
    }

var model = new MovieViewDTO();
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

public async Task<PaginatedResponseDTO<Movie>> Get(SearchMoviesParametersDTO model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);

    if(model.Releases) {
        var today = DateTime.Today;

        queryableMovies = queryableMovies
            .Where(x => x.ReleaseDate >= today);
    }

    if(model.GenreID != 0)
        queryableMovies = queryableMovies
            .Where(x => x.GenresMovie
                .Select(y => y.GenreID)
                .Contains(model.GenreID));

    if(model.MostVoted) {
        queryableMovies = queryableMovies.OrderByDescending(m => m.VotesMovies.Average(vm
```

```
=> vm.Voto));  
}  
  
    var paginatedResponse = new PaginatedResponseDTO<Movie>();  
    paginatedResponse.totalPages = await  
queryableMovies.CalculateTotalPages(model.RecordCount);  
  
    //Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida  
    paginatedResponse.Records = await queryableMovies.ToPage(model.pagination).ToListAsync();  
    return paginatedResponse;  
}  
  
public async Task<MovieUpdateDTO> PutGet(int id) {  
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.  
    var movieActionResult = await Get(id);  
  
    if(movieActionResult == null) return null;  
  
    var movieViewDTO = movieActionResult;      //Será el DTO  
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();  
    var unselectedGenres = await context.Genres  
        .Where(x => !selectedGenresIDs.Contains(x.ID))  
        .ToListAsync();  
  
    var model = new MovieUpdateDTO();  
    model.Movie = movieViewDTO.Movie;  
    model.UnselectedGenres = unselectedGenres;  
    model.SelectedGenres = movieViewDTO.Genres;  
    model.Actors = movieViewDTO.Actors;  
  
    return model;  
}  
  
public async Task<int> Post(Movie movie) {  
    if(!string.IsNullOrWhiteSpace(movie.Poster)) {  
        var poster = Convert.FromBase64String(movie.Poster);  
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);  
    }  
  
    WriteActorsOrder(movie);  
  
    context.Add(movie);  
    await context.SaveChangesAsync();  
    return movie.ID;  
}  
  
private static void WriteActorsOrder(Movie movie) {  
    if(movie.MovieActor is not null) {  
        for(int i = 0; i < movie.MovieActor.Count; i++) {  
            movie.MovieActor[i].Orden = i + 1;  
        }  
    }  
}
```

```

        }
    }
}

public async Task Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if(movieDB is null) throw new ApplicationException($"Movie {movie.ID} was not found!");

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if(!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, "jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE
}

public async Task Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if(movie is null) throw new ApplicationException($"Movie {id} was not found!");

    context.Remove(movie); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if(string.IsNullOrWhiteSpace(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

    }
}
}

```

CreateMovie.razor

```

@page "/movies/create"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject MoviesRepository repository
@inject GenresRepository genreRepository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]

```

```

<h3>Create Movie</h3>

@if(ShowForm) {
<MoviesForm Movie="Movie" OnValidSubmit="Create"
    UnselectedGenres="Unselected"/>
}
else {

}
@code {
    private Movie Movie = new Movie();
    private List<Genre> Unselected = new List<Genre>();
    public bool ShowForm { get; set; } = false;

    protected async override Task OnInitializedAsync() {
        Unselected = await genreRepository.Get();
        ShowForm = true;
    }

    async Task Create() {
        var movielD = await repository.Post(Movie);
        navManager.NavigateTo($"~/movie/{movielD}/{Movie.Title.Replace(" ", "-")}");
    }
}

```

Usamos el método **PutGet** en **OnInitializedAsync** para que retorne el modelo.

EditMovie.razor

```

@page "/movie/edit/{MovielD:int}"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject MoviesRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
<h3>Edit Movie</h3>

@if(Movie is not null) {
<MoviesForm Movie="Movie" OnValidSubmit="Edit"
    UnselectedGenres="Unselected"
    SelectedGenres="Selected"
    SelectedActors="SelectedActors"/>
}
else {
    <LoadingWheel />
}

@code {
    [Parameter] public int MovielD { get; set; }
}

```

```

private Movie? Movie;
private List<Genre> Unselected = new List<Genre>();
private List<Genre> Selected = new List<Genre>();
private List<Actor> SelectedActors = new List<Actor>();

protected override async Task OnInitializedAsync() {
    var model = await repository.PutGet(MovieID);

    Movie = model.Movie;
    Unselected = model.UnselectedGenres;
    Selected = model.SelectedGenres;
    SelectedActors = model.Actors;
}

private async Task Edit() {
    await repository.Put(Movie);
    navManager.NavigateTo($"~/movie/{MovieID}");
}
}

```

FilterMovie.razor

```

@page "/movies/filter"
@inject MoviesRepository repository
@inject GenresRepository genresRepository
@inject NavigationManager navManager
<h3>Movies filter</h3>

<div class="row g-3 align-items-center mb-3">
    <div class="col-sm-3">
        <input type="text" class="form-control" id="title" placeholder="Movie title"
            @bind-value="Title" @bind-value:event="oninput"
            @onkeypress="@(KeyboardEventArgs e) => TitleKeyPress(e)" />
    </div>
    <div class="col-sm-3">
        <select class="form-select" @bind="Genre">
            <option value="0">-- Select a genre --</option>
            @foreach(var item in genres) {
                <option value="@item.ID">@item.Name</option>
            }
        </select>
    </div>
    <div class="col-sm-6" style="display:flex;">
        <div class="form-check me-2">
            <input type="checkbox" class="form-check-input" id="premieres" @bind="futurePremieres" />
            <label class="form-check-label" for="premieres">Future premieres</label>
        </div>
        <div class="form-check me-2">
            <input type="checkbox" class="form-check-input" id="billboard" @bind="onBillboard" />
            <label class="form-check-label" for="billboard">On billboard</label>
        </div>
    </div>

```

```
</div>
<div class="form-check">
    <input type="checkbox" class="form-check-input" id="mostVoted" @bind="mostVoted" />
    <label class="form-check-label" for="mostVoted">Most voted</label>
</div>
</div>

<div class="col-12">
    <button type="button" class="btn btn-primary" @onclick="FilteredMovies">Filter</button>
    <button type="button" class="btn btn-danger" @onclick="Clean">Clean</button>
</div>
</div>

<Pagination ActualPage="actualPage" TotalPages="totalPages" SelectedPage="SelectedPage" />

<MoviesList Movies="Movies" />
```

```
@code {
    [Parameter, SupplyParameterFromQuery] public string Title { get; set; } = "";
    [Parameter, SupplyParameterFromQuery(Name = "genereid")] public int Genre { get; set; } = 0;
    private List<Genre> genres = new List<Genre>();
    [Parameter, SupplyParameterFromQuery] public bool futurePremieres { get; set; } = false;
    [Parameter, SupplyParameterFromQuery] public bool onBillboard { get; set; } = false;
    [Parameter, SupplyParameterFromQuery] public bool mostVoted { get; set; } = false;
    private List<Movie>? Movies;
    Dictionary<string, string> queryStringDict = new Dictionary<string, string>();
    [Parameter, SupplyParameterFromQuery] public int actualPage { get; set; } = 1;
    private int totalPages { get; set; } = 1;

    protected override async Task OnInitializedAsync() {
        //Movies = repository.GetMovies();
        if(actualPage == 0) actualPage = 1; //Por las dudas de que en el queryString viniera un 0.
        await GetGenres();
        await PerformFilter();
    }

    private async Task SelectedPage(int page) {
        actualPage = page;
        await FilteredMovies();
    }

    private async Task GetGenres() {
        genres = await genresRepository.Get();
    }

    private async Task TitleKeyPress(KeyboardEventArgs e)
    {
        if(e.Key == "Enter") {
            await FilteredMovies();
        }
    }
}
```

```

private async Task FilteredMovies() {
    var queryString = GenerateQueryStrings();
    navManager.NavigateTo($"/movies/filter?{queryString}");
    await PerformFilter();
}

private async Task PerformFilter() {
    var searchParameters = GenerateSearchParameters();
    var paginatedResponse = await repository.Get(searchParameters);
    Movies = paginatedResponse.Records;
    totalPages = paginatedResponse.totalPages;
}

private SearchMoviesParametersDTO GenerateSearchParameters() {
    var response = new SearchMoviesParametersDTO();
    response.GenreID = Genre;
    response.Title = Title;
    response.Onbillboard = onBillboard;
    response.Releases = futurePremieres;
    response.MostVoted = mostVoted;
    response.Page = actualPage;
    return response;
}

private string GenerateQueryStrings() {
    if(queryStringDict == null)
        queryStringDict = new Dictionary<string, string>();

    queryStringDict["genreid"] = Genre.ToString();
    queryStringDict["title"] = Title ?? string.Empty;
    queryStringDict["futurePremieres"] = futurePremieres.ToString();
    queryStringDict["onBillboard"] = onBillboard.ToString();
    queryStringDict["mostVoted"] = mostVoted.ToString();
    queryStringDict["page"] = actualPage.ToString();

    var defaultValues = new List<string>() { "false", "", "0" };

    return string.Join("&", queryStringDict.Where(x =>
        !defaultValues.Contains(x.Value.ToLower()))
        .Select(x => $"{x.Key}={System.Web.HttpUtility.UrlEncode(x.Value)}")
        .ToArray());
}

private async Task Clean() {
    Title = "";
    Genre = 0;
    futurePremieres = false;
    onBillboard = false;
    mostVoted = false;
}

```

```
    await FilteredMovies();

}
```

MovieForm.razor

```
@inject ActorsRepository repository

<EditForm Model="Movie" OnValidSubmit="OnDataAnnotationsValidated">
    <DataAnnotationsValidator />

    <div class="mb-3">
        <label>Title:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Movie.Title" />
            <ValidationMessage For="@(() => Movie.Title)" />
        </div>
    </div>

    <div class="mb-3">
        <label>On billboard:</label>
        <div>
            <InputCheckbox @bind-Value="@Movie.OnBillboard" />
            <ValidationMessage For="@(() => Movie.OnBillboard)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Trailer:</label>
        <div>
            <InputText class="form-control" @bind-Value="@Movie.Trailer" />
            <ValidationMessage For="@(() => Movie.Trailer)" />
        </div>
    </div>

    <div class="mb-3">
        <label>Release date:</label>
        <div>
            <InputDate class="form-control" @bind-Value="@Movie.ReleaseDate" />
            <ValidationMessage For="@(() => Movie.ReleaseDate)" />
        </div>
    </div>

    <div class="mb-3">
        <InputImg Label="Poster" SelectedImage="SelectedImage" ImageURL="@ImageURL" />
    </div>

    <div class="mb-3 form-markdown">
        <InputMD @bind-Value="@Movie.Summary" />
    </div>
```

```

For=@(() => Movie.Summary)
Label="Summary" />
</div>

<div class="mb-3">
    <label>Genres:</label>
    <div>
        <MultipleSelector Selected="Selected" Unselected="Unselected"></MultipleSelector>
    </div>
</div>

<div class="mb-3">
    <label>Actors:</label>
    <div>
        <MultipleSelectorTypeahead Context="Actor" SearchMethod="SearchActors"
            SelectedElements="SelectedActors" NotFoundTemplate="No actors were found">
            <ListTemplate>
                @Actor.Name / <input type="text" placeholder="Character" @bind="Actor.Character" />
            </ListTemplate>
            <ResultTemplate>
                
                @Actor.Name
            </ResultTemplate>
        </MultipleSelectorTypeahead>
    </div>
</div>

<button class="btn btn-success" type="submit">Save changes</button>
</EditForm>
@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;

    [Parameter]
    [EditorRequired]
    public EventCallback OnValidSubmit { get; set; }

    [Parameter]
    public List<Genre> SelectedGenres { get; set; } = new List<Genre>();
    [Parameter]
    [EditorRequired]
    public List<Genre> UnselectedGenres { get; set; } = new List<Genre>();

    [Parameter]
    public List<Actor> SelectedActors { get; set; } = new List<Actor>();

    private List<MultipleSelectorModel> Selected { get; set; } = new List<MultipleSelectorModel>();
    private List<MultipleSelectorModel> Unselected { get; set; } = new List<MultipleSelectorModel>();
}

```

```

string? ImageURL;

protected override void OnInitialized() {
    if(!string.IsNullOrEmpty(Movie.Poster)) {
        ImageURL = Movie.Poster;
        Movie.Poster = null;      //Al editar, seteamos el URL y limpiamos el dato.
                                //Si no se le carga uno nuevo, no será re-enviado.
    }

    Selected = SelectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(), x.Name)).ToList();
    Unselected = UnselectedGenres.Select(x => new MultipleSelectorModel(x.ID.ToString(),
x.Name)).ToList();
}

private void SelectedImage(string imageBase64) {
    Movie.Poster = imageBase64;
    ImageURL = null;
}

private async Task<IEnumerable<Actor>> SearchActors(string searchText) {
    return await repository.Get(searchText);
}

private async Task OnDataAnnotationsValidated() {
    //Obtenemos el listado de géneros seleccionados recuperando los IDs (Key).
    Movie.GenresMovie = Selected
        .Select(x => new GenresMovie { GenreID = int.Parse(x.Key) }).ToList();

    //Obtenemos el listado de actores seleccionados recuperando los IDs (ID) y los personajes de c/u.
    Movie.MovieActor = SelectedActors
        .Select(x => new MovieActor { ActorID = x.ID, Character = x.Character }).ToList();

    await OnValidSubmit.InvokeAsync();
}
}

```

ViewMovie.razor

```

@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
@inject MoviesRepository repository
@inject SweetAlertService swal

@if(model is null) {
    <LoadingWheel />
}
else {
    <h2>@movie.Title (@movie.ReleaseDate!.Value.ToString("yyyy"))</h2>
    @foreach(var genre in model.Genres) {
        <a class="me-2 badge bg-primary rounded-pill text-decoration-none"
        href="movies/filter?genreid=@genre.ID">@genre.Name</a>
    }
}

```

```

}

<span>| @movie.ReleaseDate!.Value.ToString("dd MM yyyy")
| Media: @model.VotesMedia.ToString("0.#")/5
| Your vote: <Rating MaxPoints="5" SelectedPoint="model.UserVote"
OnRating="OnRating"></Rating></span>

<div class="d-flex mt-2">
    <span style="display:inline-block;" class="me-2">
        
    </span>

    <iframe width="560" height="315" src="https://www.youtube.com/embed/@movie.Trailer"
title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
</div>
<div class="mt-2">
    <h3>Summary</h3>
    <div>
        <ShowMD MDContent="@movie.Summary" />
    </div>
</div>

<div class="mt-2">
    <h3>Actors</h3>
    <div class="d-flex flex-column">
        @foreach(var actor in model.Actors) {
            <div class="mb-2">
                
                <span style="display:inline-block;width:200px;">@actor.Name</span>
                <span style="display:inline-block;width:45px;">...</span>
                <span>@actor.Character</span>
            </div>
        }
    </div>
</div>
}

@code {
    [Parameter] public int MovieID { get; set; }
    [Parameter] public string MovieName { get; set; } = null!;
    private MovieViewDTO? model;
    private Movie movie = null!;

    protected override async Task OnInitializedAsync() {
        model = await repository.Get(MovieID);
        movie = model!.Movie;
    }

    private async Task OnRating(int selectedVote) {
}

```

```
//model!.UserVote = selectedVote;
//var voteMovieDTO = new VoteMovieDTO() {
//  MovieID = MovieID,
//  Voto = selectedVote
//};

//var responseHTTP = await repository.Post("api/votes", voteMovieDTO);

//if(responseHTTP.Error) {
//  var ErrMessage = await responseHTTP.GetErrMessage();
//  await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
//}
//else
//  await swal.FireAsync("Success!", "Your vote has been received.", SweetAlertIcon.Success);
}
}
```

MoviesList.razor

```
@inject IJSRuntime js
@inject MoviesRepository repository
@inject SweetAlertService swal

<div style="display:flex;flex-wrap:wrap;align-items:center;">
<GenericList List="Movies">
  <Loading>
    @Loading
  </Loading>
  <NoRecords>
    @NoRecords
  </NoRecords>
  <HasRecords Context="movie">
    <MovieItem Movie="movie"
      DeleteMovie="DeleteMovie"
      @key="movie.ID"/>
  </HasRecords>
</GenericList>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public List<Movie>? Movies { get; set; }
    [Parameter]
    public RenderFragment Loading { get; set; } = null!;
    [Parameter]
    public RenderFragment NoRecords { get; set; } = null!;

    private async Task DeleteMovie(Movie movie) {
        var confirmed = await js.Confirm($"Do you want to delete the movie '{movie.Title}'?");
    }
}
```

```
if (confirmed) {
    await repository.Delete(movie.ID);
    Movies!.Remove(movie);
}
}
```

MovieItem.razor

```
<div class="me-2 mb-2" style="text-align:center">
    <a href="@urlViewMovie">
        
    </a>
    <p style="max-width:225px;height:44px;font-size:15px;font-weight:bold;">
        <a href="@urlViewMovie" class="text-decoration-none">@Movie.Title</a>
    </p>
    <AuthorizeView Roles="admin">
        <div>
            <a class="btn btn-info" href="@urlEditMovie">Edit</a>
            <button type="button" class="btn btn-danger"
                    @onclick="@(() => DeleteMovie.InvokeAsync(Movie))">Delete</button>
        </div>
    </AuthorizeView>
</div>

@code {
    [Parameter]
    [EditorRequired]
    public Movie Movie { get; set; } = null!;
    [Parameter]
    public bool ShowButtons { get; set; } = false;
    [Parameter]
    public EventCallback<Movie> DeleteMovie { get; set; }

    private string urlViewMovie = string.Empty;
    private string urlEditMovie = string.Empty;
    protected override void OnInitialized()
    {
        urlViewMovie = $"~/movie/{Movie.ID}/{Movie.urlTitle()}";
        urlEditMovie = $"~/movie/edit/{Movie.ID}";
    }
}
```

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
```

```
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;
using BlazorPeliculasServer.Helpers;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new
InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationContext>(options =>
//    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
    .AddRoles<IdentityRole>()
    .AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();
builder.Services.AddTransient<GenresRepository>();
builder.Services.AddTransient<ActorsRepository>();
builder.Services.AddTransient<MoviesRepository>();
builder.Services.AddScoped<IFileSaver, FileSaverLocal>();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddTransient<AuthenticationStateService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Copiamos el Código de **Index.razor** del proyecto anterior en el **Index.razor** de este proyecto y corregimos:

Logout.razor

```
@page "/"
@inject MoviesRepository repository

<PageTitle>Blazor Movies</PageTitle>
<div>
    <h3>On billboard</h3>
    <div>
        <MoviesList Movies="OnBoard">
            <Loading>
                <LoadingWheel />
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                <LoadingWheel />
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>
```

```
@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }

    protected override async Task OnInitializedAsync() {
        var homePage = await repository.Get();
        OnBoard = homePage.OnBoard;
        NextReleases = homePage.NextReleases;
    }
}
```

Agregamos los estilos en el archivo **custom.css**.

```
custom.css
.multiple-selector {
    display:flex;
}

.selectable-ul {
    height:200px;
    overflow-y:auto;
    list-style-type:none;
    width:170px;
    padding:0;
    border-radius:3px;
    border:1px solid #ccc;
}

.selectable-ul li {
    cursor:pointer;
    border-bottom:1px #eee solid;
    padding:2px 10px;
    font-size:14px;
}

.selectable-ul li:hover {
    background-color:#08c;
}

.multiple-selector-buttons {
    display:flex;
    flex-direction:column;
    justify-content:center;
    padding:5px;
}

.multiple-selector-buttons button {
    margin:5px;
```

```
}
```

```
.checked {
    color:orange;
}
```

Migrando votación

Creamos la clase **VotesRepository.cs** en la carpeta **Repositories**. Copiamos en ella el código de **VotesControllers.cs** del proyecto anterior.

Eliminamos los atributos **HttpGet**, **HttpPost**, **AllowAnonymous**, etc. Traemos los namespaces de **Microsoft.AspNetCore.Mvc** y **Microsoft.EntityFrameworkCore**. Eliminamos los **ActionResult**, **FromQuery** y **NoContent()**.

VotesRepository.cs

```
using AutoMapper;
using BlazorPeliculasServer.Data;
using BlazorPeliculasServer.DTOs;
using BlazorPeliculasServer.Entities;
using BlazorPeliculasServer.Helpers;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
```

```
namespace BlazorPeliculasServer.Repositories {
    public class VotesRepository {
        private readonly ApplicationDbContext context;
        private readonly UserManager<IdentityUser> userManager;
        private readonly IMapper mapper;
        private readonly AuthenticationStateService authenticationStateService;

        public VotesRepository(ApplicationDbContext context,
            UserManager<IdentityUser> userManager,
            IMapper mapper,
            AuthenticationStateService authenticationStateService) {
            this.context = context;
            this.userManager = userManager;
            this.mapper = mapper;
            this.authenticationStateService = authenticationStateService;
        }

        public async Task Vote(VoteMovieDTO voteMovieDTO) {
            //Tomo el nombre (email) del usuario y con eso busco y obtengo el registro de dicho usuario.
            var userID = await authenticationStateService.GetCurrentUserID();

            if(userID is null) {
                //No debería pasar nunca, pero por las dudas...
                return;
            }
        }
    }
}
```

```
var currentVote = await context.VotesMovies
    .FirstOrDefaultAsync(x => x.MovieID == voteMovieDTO.MovieID && x.UserID == userID);

if(currentVote is null) {
    //Aún no voto. Crear uno nuevo.
    var voteMovie = mapper.Map<VoteMovie>(voteMovieDTO);
    voteMovie.UserID = userID;
    voteMovie.VoteWhen = DateTime.Now;
    context.Add(voteMovie);    //Marcar para grabar
}
else {
    //Ya votó. Actualizar la fecha y el voto
    currentVote.VoteWhen = DateTime.Now;
    currentVote.Voto = voteMovieDTO.Voto;
}

await context.SaveChangesAsync();
}
```

Program.cs

```
using BlazorPeliculasServer.Areas.Identity;
using BlazorPeliculasServer.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.UI;
using Microsoft.EntityFrameworkCore;
using CurrieTechnologies.Razor.SweetAlert2;
using BlazorPeliculasServer.Repositories;
using BlazorPeliculasServer.Helpers;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
var connectionString = builder.Configuration.GetConnectionString("DefaultConnection") ?? throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

var optionsBuilder = new DbContextOptionsBuilder<ApplicationContext>();
optionsBuilder.UseSqlServer(connectionString);
builder.Services.AddTransient(_ => new ApplicationContext(optionsBuilder.Options));

//builder.Services.AddDbContext<ApplicationContext>(options =>
//    options.UseSqlServer(connectionString));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddDefaultIdentity<IdentityUser>(options => options.SignIn.RequireConfirmedAccount
= false)
```

```
.AddRoles<IdentityRole>()
.AddEntityFrameworkStores<ApplicationContext>();
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddScoped<AuthenticationStateProvider,
RevalidatingIdentityAuthenticationStateProvider<IdentityUser>>();
builder.Services.AddSingleton<WeatherForecastService>();
builder.Services.AddSweetAlert2();
builder.Services.AddTransient<UsersRepository>();
builder.Services.AddTransient<GenresRepository>();
builder.Services.AddTransient<ActorsRepository>();
builder.Services.AddTransient<MoviesRepository>();
builder.Services.AddTransient<VotesRepository>();
builder.Services.AddScoped<IFileSaver, FileSaverLocal>();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddTransient<AuthenticationStateService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    app.UseMigrationsEndPoint();
}
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.UseAuthorization();

app.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

ViewMovie.razor

```
@page "/movie/{MovieID:int}"
@page "/movie/{MovieID:int}/{MovieName}"
@inject MoviesRepository repository
@inject VotesRepository votesRepository
```

```

@inject SweetAlertService swal

@if(model == null) {
    <LoadingWheel />
}
else {
    <h2>@movie.Title (@movie.ReleaseDate!.Value.ToString("yyyy"))</h2>
    @foreach(var genre in model.Genres) {
        <a class="me-2 badge bg-primary rounded-pill text-decoration-none"
        href="movies/filter?genreid=@genre.ID">@genre.Name</a>
    }

    <span>| @movie.ReleaseDate!.Value.ToString("dd MM yyyy")
    | Media: @model.VotesMedia.ToString("0.#")/5
    | Your vote: <Rating MaxPoints="5" SelectedPoint="model.UserVote"
    OnRating="OnRating"></Rating></span>

    <div class="d-flex mt-2">
        <span style="display:inline-block;" class="me-2">
            
        </span>

        <iframe width="560" height="315" src="https://www.youtube.com/embed/@movie.Trailer"
        title="YouTube video player" frameborder="0" allow="accelerometer; autoplay; clipboard-write;
        encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen></iframe>
    </div>
    <div class="mt-2">
        <h3>Summary</h3>
        <div>
            <ShowMD MDContent="@movie.Summary" />
        </div>
    </div>

    <div class="mt-2">
        <h3>Actors</h3>
        <div class="d-flex flex-column">
            @foreach(var actor in model.Actors) {
                <div class="mb-2">
                    
                    <span style="display:inline-block;width:200px;">@actor.Name</span>
                    <span style="display:inline-block;width:45px;">...</span>
                    <span>@actor.Character</span>
                </div>
            }
        </div>
    </div>
}

@code {
    [Parameter] public int MovieID { get; set; }
}

```

```
[Parameter] public string MovieName { get; set; } = null!;
private MovieViewDTO? model;
private Movie movie = null!;

protected override async Task OnInitializedAsync() {
    model = await repository.Get(MovieID);
    movie = model!.Movie;
}

private async Task OnRating(int selectedVote) {
    model!.UserVote = selectedVote;
    var voteMovieDTO = new VoteMovieDTO() {
        MovieID = MovieID,
        Voto = selectedVote
    };

    await votesRepository.Vote(voteMovieDTO);
    await swal.FireAsync("Success!", "Your vote has been received.", SweetAlertIcon.Success);
}
}
```

Resumen



Características principales de una aplicación de Blazor del lado del servidor



El sistema de usuarios es bastante más sencillo de realizar



Realizamos una migración de nuestra aplicación



Publicamos nuestra aplicación de Blazor del lado del servidor hacia un Azure App Service

Internacionalización

Crearemos una app webassembly multi-idioma.

La internacionalización la podemos separar en dos procesos: globalización y localización.

Globalización

Es el proceso de diseñar aplicaciones que soporten distintas culturas. Así, nosotros somos los que tenemos que preparar nuestra aplicación para que sea multi idioma o para que muestre datos como fechas y números de acuerdo con la cultura del usuario.

Localización

Se refiere al proceso de adaptar una aplicación globalizada a una cultura específica. Por ejemplo, digamos que tu aplicación soporta dos idiomas español e inglés, pero en términos de cultura solamente soporta la cultura de Estados Unidos.

Si un usuario indica que quiere la aplicación en español de República Dominicana, el proceso de localización va a determinar cómo servir la aplicación.

Si en inglés con cultura americana o en español con cultura americana.

Es decir, la localización se encarga de seleccionar la mejor manera de servir la aplicación al usuario según los recursos disponibles.

En **.NET** encapsulamos la información de idioma y región en la clase `CultureInfo`.

Así, si queremos indicar que queremos utilizar el idioma español y la región de Estados Unidos, podemos decir **new CultureInfo("es-US")**, donde **es** viene de español y **US** es de Estados Unidos.

Si no queremos especificar una región y solamente queremos especificar el idioma, podemos hacerlo así:

new CultureInfo("es")

En este caso, la región escogida es la región por defecto del idioma. En el caso del español, esta región es España.

Si vamos a tener una aplicación multi idiomas, entonces tenemos que crear al menos un archivo por idioma donde colocaremos el texto de la aplicación en un idioma particular.

Eso lo hacemos en archivos de recursos cuya extensión es **resx**. Esto se representa en un diccionario de llaves y valores, donde el valor es el texto traducido.

Por ejemplo, en la siguiente imagen tenemos dos ejemplos de archivo de recursos **resx**, donde cada uno tiene tres columnas.

Archivos de Recursos

Archivo - Inglés

	Name	Value	Comment
▶	Greeting	Hello, world!	
	index.welcome	Welcome to your new app.	
	required	This field is required	
*			

Archivo - Español

	Name	Value	Comment
▶	Greeting	Hola, mundo!	
	index.welcome	Bienvenido a tu nueva aplicación	
	required	Este campo es requerido	
*			

La primera columna es la llave. La segunda es el valor traducido y la tercera es una columna de comentarios.

En la primera columna de ambos archivos tenemos **Greeting**. Mientras que en el archivo de inglés en la segunda columna tenemos **Hello, world!** en el archivo en español tenemos **Hola, mundo!**.

Luego en nuestros componentes o clases de C#, en general podemos utilizar **IStringLocalizer** para así colocar el texto según el idioma del usuario.

Culture y Culture UI

Existen dos valores de cultura que podemos utilizar en este ASP.NET Core.: **Culture** y **Culture UI**.

Culture

Con éste configuramos los resultados de funciones que dependen de culturas. Por ejemplo, si mostramos la fecha actual en la pantalla, la cultura de la fecha se mostrará según la cultura utilizada.

Si tenemos la cultura de Estados Unidos se va a mostrar mes, día, año, y si tenemos la cultura, por ejemplo, de Argentina, veremos la fecha en formato día, mes, año.

Culture UI

Con éste indicamos los archivos de recursos a utilizar para la aplicación. Visto de algún modo con **Culture** podemos definir la cultura de los datos de la aplicación como fechas y números. Mientras que con **Culture UI** definimos el idioma de la aplicación.

Esto es interesante porque nos permite separar la parte de la cultura de los datos con el idioma.

Así, si por ejemplo, queremos que toda la aplicación utilice una cultura específica, como por ejemplo la de Estados Unidos pero, si te permite traducir la aplicación a idiomas como español, inglés, francés, ruso, etc... podremos hacerlo sin ningún problema, porque por un lado tenemos **Culture** y por el otro tenemos **Culture UI**.

Existen tres proveedores de cultura por defecto que son **QueryStrings**, **cookies** y **cabecera HTTP**.

QueryStrings

`http://localhost:5000/?culture=es-DO&ui-culture=es-DO`

Cookies

Guardamos una cookie en el navegador del usuario:

`c=en-UK | uic=en-US`

Como podemos ver aquí lo podemos tener uno en una cultura y otro en otra cultura, y no hay ningún problema.

Esto lo vamos a utilizar en Blazor server.

HTTP Header

Finalmente tenemos la opción de la cabecera HTTP. A través de la cabecera `Accept-Language` Podemos configurar la cultura a utilizar.

`Cabecera HTTP:Accept-Language`

Este valor es enviado automáticamente por los navegadores según el idioma del usuario.

App multi-idioma

Crearemos un nuevo proyecto del tipo **Blazor Webassembly App** que llamaremos **BlazorWebAssemblyIdiomas** con ASP.NET Core hosted habilitado.

Para que sea multi-idioma, lo primero es instalar el paquete **NuGet** llamado **Microsoft.Extensions.Localization**. Luego es necesario configurar la localización en la clase **Program.cs** del proyecto **Client**.

Program.cs

```
using BlazorWebAssemblyIdiomas.Client;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

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

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

Creamos una nueva carpeta del proyecto **Client** llamada **Resources** y agregamos un nuevo archivo del tipo **Resource** que llamaremos **Resource.resx**. Este es el archivo por defecto. Siempre, por defecto, es el de idioma inglés. Por lo tanto, agregamos otro archivo que llamaremos **Resource.es.resx**.

Ambos archivos se verían así:

Resource.resx

```
<?xml version="1.0" encoding="utf-8"?>
<root>
  <!--
    Microsoft ResX Schema

    Version 1.3

    The primary goals of this format is to allow a simple XML format
    that is mostly human readable. The generation and parsing of the
    various data types are done through the TypeConverter classes
    associated with the data types.

    Example:

    ... ado.net/XML headers & schema ...
    <resheader name="resmimetype">text/microsoft-resx</resheader>
    <resheader name="version">1.3</resheader>
    <resheader name="reader">System.Resources.ResXResourceReader,
    System.Windows.Forms, ...</resheader>
    <resheader name="writer">System.Resources.ResXResourceWriter,
    System.Windows.Forms, ...</resheader>
```

```
<data name="Name1">this is my long string</data>
<data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
<data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
    [base64 mime encoded serialized .NET Framework object]
</data>
<data name="Icon1" type="System.Drawing.Icon, System.Drawing"
mimetype="application/x-microsoft.net.object.bytearray.base64">
    [base64 mime encoded string representing a byte array form of the .NET
Framework object]
</data>
```

There are any number of "resheader" rows that contain simple name/value pairs.

Each data row contains a name, and value. The row also contains a type or mimetype. Type corresponds to a .NET class that support text/value conversion through the TypeConverter architecture. Classes that don't support this are serialized and stored with the mimetype set.

The mimetype is used for serialized objects, and tells the ResXResourceReader how to depersist the object. This is currently not extensible. For a given mimetype the value must be set accordingly:

Note - application/x-microsoft.net.object.binary.base64 is the format that the ResXResourceWriter will generate, however the reader can read any of the formats listed below.

mimetype: application/x-microsoft.net.object.binary.base64
value : The object must be serialized with
 : System.Serialization.Formatters.Binary.BinaryFormatter
 : and then encoded with base64 encoding.

mimetype: application/x-microsoft.net.object.soap.base64
value : The object must be serialized with
 : System.Runtime.Serialization.Formatters.Soap.SoapFormatter
 : and then encoded with base64 encoding.

mimetype: application/x-microsoft.net.object.bytearray.base64
value : The object must be serialized into a byte array
 : using a System.ComponentModel.TypeConverter
 : and then encoded with base64 encoding.

-->

```
<xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xsd:element name="root" msdata:lsDataSet="true">
        <xsd:complexType>
            <xsd:choice maxOccurs="unbounded">
                <xsd:element name="data">
```

```
<xsd:complexType>
  <xsd:sequence>
    <xsd:element name="value"
      type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
    <xsd:element name="comment"
      type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
    </xsd:sequence>
    <xsd:attribute name="name"
      type="xsd:string" msdata:Ordinal="1" />
    <xsd:attribute name="type" type="xsd:string"
      msdata:Ordinal="3" />
    <xsd:attribute name="mimetype"
      type="xsd:string" msdata:Ordinal="4" />
  </xsd:complexType>
</xsd:element>
<xsd:element name="resheader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value"
        type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
      </xsd:sequence>
      <xsd:attribute name="name"
        type="xsd:string" use="required" />
    </xsd:complexType>
  </xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<resheader name="resmimetype">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
  <value>1.3</value>
</resheader>
<resheader name="reader">
  <value>System.Resources.ResXResourceReader, System.Windows.Forms,
Version=2.0.3500.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
  <value>System.Resources.ResXResourceWriter, System.Windows.Forms,
Version=2.0.3500.0, Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
</root>
```

En el archivo por defecto (el de inglés) tenemos que cambiar el Access Modifier a **Public** para que genere una clase de C# que podremos referenciar.

Luego de grabar, se lo ve así:

Resource.resx

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<root>
```

```
<!--
```

```
Microsoft ResX Schema
```

```
Version 2.0
```

The primary goals of this format is to allow a simple XML format that is mostly human readable. The generation and parsing of the various data types are done through the TypeConverter classes associated with the data types.

Example:

```
... ado.net/XML headers & schema ...
<resheader name="resmimetype">text/microsoft-resx</resheader>
<resheader name="version">2.0</resheader>
<resheader name="reader">System.Resources.ResXResourceReader, System.Windows.Forms,
...</resheader>
<resheader name="writer">System.Resources.ResXResourceWriter, System.Windows.Forms,
...</resheader>
<data name="Name1"><value>this is my long string</value><comment>this is a
comment</comment></data>
<data name="Color1" type="System.Drawing.Color, System.Drawing">Blue</data>
<data name="Bitmap1" mimetype="application/x-microsoft.net.object.binary.base64">
    <value>[base64 mime encoded serialized .NET Framework object]</value>
</data>
<data name="Icon1" type="System.Drawing.Icon, System.Drawing" mimetype="application/x-
microsoft.net.object.bytearray.base64">
    <value>[base64 mime encoded string representing a byte array form of the .NET Framework
object]</value>
    <comment>This is a comment</comment>
</data>
```

There are any number of "resheader" rows that contain simple name/value pairs.

Each data row contains a name, and value. The row also contains a type or mimetype. Type corresponds to a .NET class that support text/value conversion through the TypeConverter architecture. Classes that don't support this are serialized and stored with the mimetype set.

The mimetype is used for serialized objects, and tells the ResXResourceReader how to depersist the object. This is currently not extensible. For a given mimetype the value must be set accordingly:

Note - application/x-microsoft.net.object.binary.base64 is the format that the ResXResourceWriter will generate, however the reader can

read any of the formats listed below.

mimetype: application/x-microsoft.net.object.binary.base64

value : The object must be serialized with

: System.Runtime.Serialization.Formatters.Binary.BinayFormatter

: and then encoded with base64 encoding.

mimetype: application/x-microsoft.net.object.soap.base64

value : The object must be serialized with

: System.Runtime.Serialization.Formatters.Soap.SoapFormatter

: and then encoded with base64 encoding.

mimetype: application/x-microsoft.net.object.bytearray.base64

value : The object must be serialized into a byte array

: using a System.ComponentModel.TypeConverter

: and then encoded with base64 encoding.

-->

```
<xsd:schema id="root" xmlns="" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
  <xsd:import namespace="http://www.w3.org/XML/1998/namespace" />
  <xsd:element name="root" msdata:lsDataSet="true">
    <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
        <xsd:element name="metadata">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="value" type="xsd:string" minOccurs="0" />
            </xsd:sequence>
            <xsd:attribute name="name" use="required" type="xsd:string" />
            <xsd:attribute name="type" type="xsd:string" />
            <xsd:attribute name="mimetype" type="xsd:string" />
            <xsd:attribute ref="xml:space" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="assembly">
          <xsd:complexType>
            <xsd:attribute name="alias" type="xsd:string" />
            <xsd:attribute name="name" type="xsd:string" />
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="data">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
              <xsd:element name="comment" type="xsd:string" minOccurs="0" msdata:Ordinal="2" />
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:string" use="required" msdata:Ordinal="1" />
            <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
            <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
            <xsd:attribute ref="xml:space" />
          </xsd:complexType>
        </xsd:element>
      </xsd:choice>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
</xsd:complexType>
</xsd:element>
<xsd:element name="resheader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:Ordinal="1" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
</xsd:schema>
<resheader name="resmimetype">
  <value>text/microsoft-resx</value>
</resheader>
<resheader name="version">
  <value>2.0</value>
</resheader>
<resheader name="reader">
  <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<resheader name="writer">
  <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089</value>
</resheader>
<data name="String1" xml:space="preserve">
  <value />
</data>
</root>
```

Al grabar el de español, también luce igual al anterior. Por lo que la modificación del **Access Modifier** no se guarda en el xml.

Traduciremos ambos textos del **Index.razor**.

Una de las formas es usar el mismo texto como llave en los lenguajes alternativos. La clase de traducción si está en el idioma por defecto usaría la llave. Si estuviera en alguno de los alternativos usaría el valor correspondiente. Realmente, iría al resource de inglés y al detectar que no hay una llave con ese texto usará el valor por defecto.

Por ej:

Resource.es.resx*		Add Resource	Remove Resource	Access Modifier:	No code gen!
	Name	Value			
...	Hello, world!	Hola, mundo!			
*					

La única diferencia entre los idiomas es que en el de español está la traducción. Por lo tanto, al buscar la clave `Hello, world!` en el de inglés no encontraría un valor y usaría la misma clave como valor por defecto.

```

C:\Users\EFM\source\repos\BlazorWebAssembly\idiomas\Client\Resources\Resource.es.resx
23/02/2023 09:17:02 5,889 bytes Everything Else UTF-8 BOM PC
<xsd:complexType>
  <xsd:attribute name="alias" type="xsd:string" />
  <xsd:attribute name="name" type="xsd:string" />
</xsd:complexType>
<xsd:element>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:>
        <xsd:attribute name="comment" type="xsd:string" minOccurs="0" msdat>
        <xsd:attribute name="name" type="xsd:string" use="required" msdata:>
          <xsd:attribute name="type" type="xsd:string" msdata:Ordinal="3" />
          <xsd:attribute name="mimetype" type="xsd:string" msdata:Ordinal="4" />
          <xsd:attribute ref="xml:space" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:element>
</xsd:complexType>
<xsd:element name="resheader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:complexType>
          <xsd:element>
            <xsd:choice>
              <xsd:complexType>
                <xsd:element>
                  <xsd:schema>
                    <resheader name="resimimetype">
                      <value>text/microsoft-resx</value>
                    </resheader>
                    <resheader name="version">
                      <value>2.0</value>
                    </resheader>
                    <resheader name="reader">
                      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0</value>
                    </resheader>
                    <resheader name="writer">
                      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0</value>
                    </resheader>
                    <data name="String1" xml:space="preserve">
                      <value />
                    </data>
                  </root>
                </xsd:element>
              </xsd:choice>
            </xsd:complexType>
          </xsd:element>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:element>
</xsd:complexType>
<xsd:element name="resheader">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="value" type="xsd:string" minOccurs="0" msdata:>
        <xsd:attribute name="name" type="xsd:string" use="required" />
        <xsd:complexType>
          <xsd:element>
            <xsd:choice>
              <xsd:complexType>
                <xsd:element>
                  <xsd:schema>
                    <resheader name="resimimetype">
                      <value>text/microsoft-resx</value>
                    </resheader>
                    <resheader name="version">
                      <value>2.0</value>
                    </resheader>
                    <resheader name="reader">
                      <value>System.Resources.ResXResourceReader, System.Windows.Forms, Version=4.0</value>
                    </resheader>
                    <resheader name="writer">
                      <value>System.Resources.ResXResourceWriter, System.Windows.Forms, Version=4.0</value>
                    </resheader>
                    <data name="Hello, world!" xml:space="preserve">
                      <value>Hola, mundo!</value>
                    </data>
                  </root>
                </xsd:element>
              </xsd:choice>
            </xsd:complexType>
          </xsd:element>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:element>
</xsd:complexType>

```

Agregamos los `using` en el `_Imports.razor`:

```

_Imports.razor
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http

```

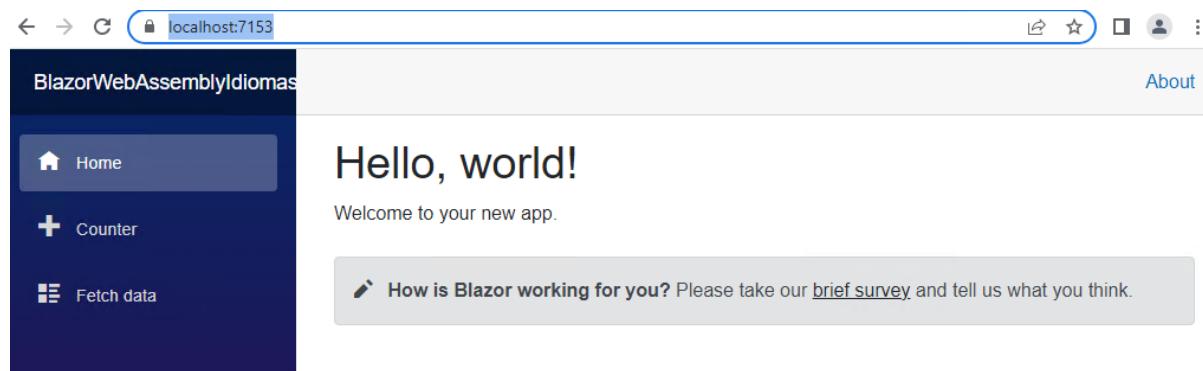
```
@using Microsoft.JSInterop  
@using BlazorWebAssemblyIdiomas.Client  
@using BlazorWebAssemblyIdiomas.Client.Shared  
@using Microsoft.Extensions.Localization  
@using BlazorWebAssemblyIdiomas.Client.Resources
```

Con eso ya Podemos comenzar a traducir nuestra app:

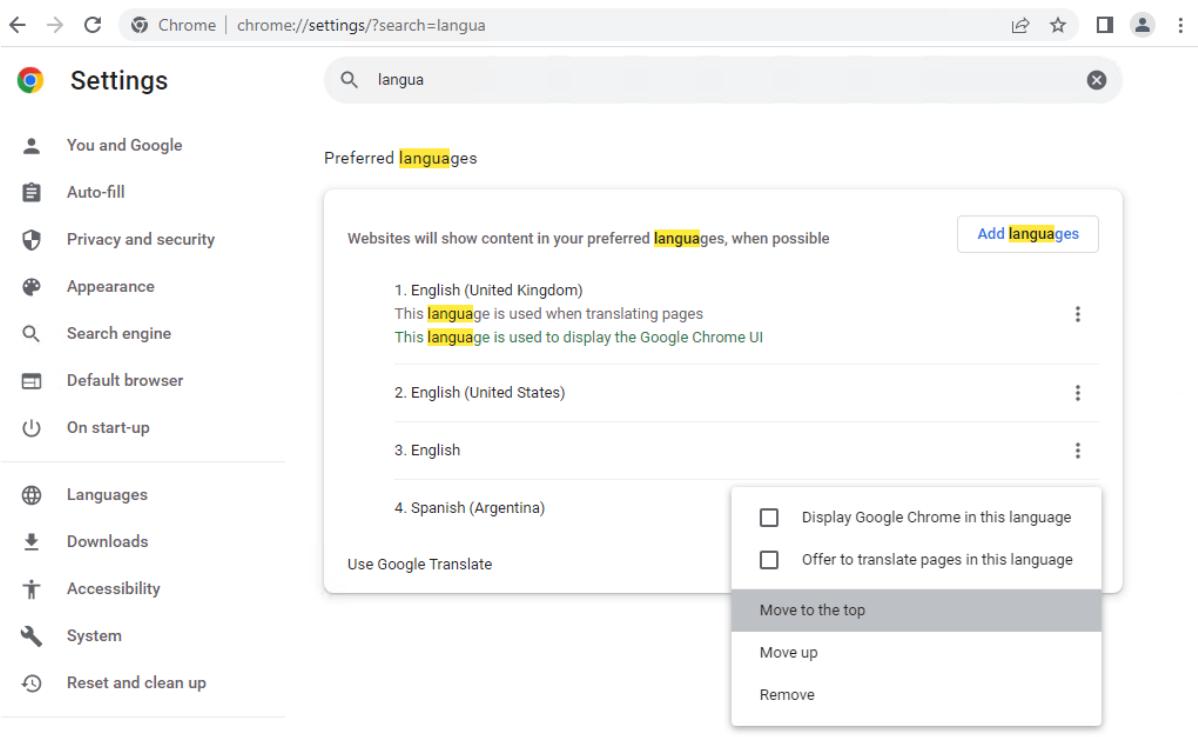
Index.razor

```
@page "/"  
@inject IStringLocalizer<Resource> localizer  
  
<PageTitle>Index</PageTitle>  
  
<h1>@localizer["Hello, world!"]</h1>  
  
Welcome to your new app.  
  
<SurveyPrompt Title="How is Blazor working for you?" />
```

Corremos la app y vemos que está en inglés porque nuestro Chrome está configurado en ese idioma:



Vamos a los settings y vemos que el primero es inglés. Elegimos Español y lo mandamos arriba de todo:



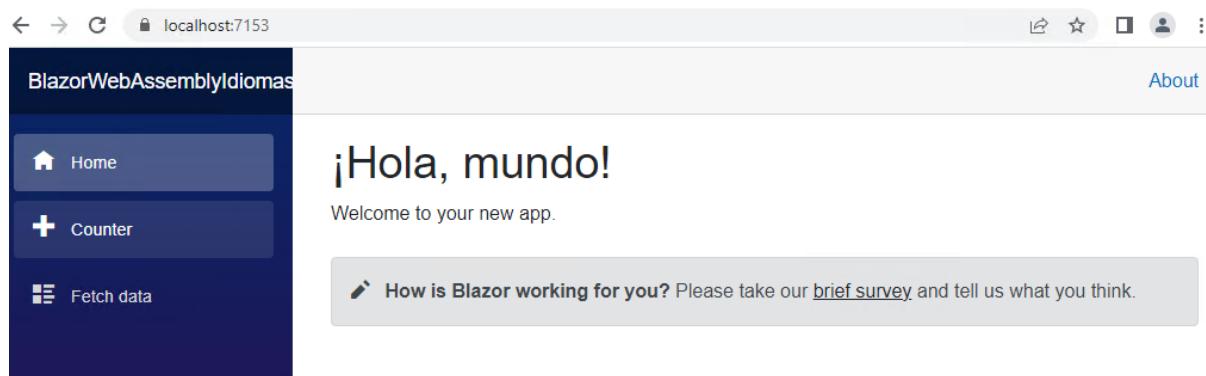
The screenshot shows the 'Languages' section in the Google Chrome settings. The search bar at the top has 'langua' typed into it. On the left, there's a sidebar with various settings categories. The main area displays a list of preferred languages:

- 1. English (United Kingdom)
This language is used when translating pages
This language is used to display the Google Chrome UI
- 2. English (United States)
- 3. English
- 4. Spanish (Argentina)

Below this list, there are two checkboxes:
 Display Google Chrome in this language
 Offer to translate pages in this language

A context menu is open over the fourth item (Spanish (Argentina)), with options: Move to the top (highlighted), Move up, and Remove.

Al refrescar, el texto aparece traducido:



The screenshot shows a Blazor application running at localhost:7153. The navigation bar at the top says 'BlazorWebAssemblyIdiomas'. The main content area displays the text '¡Hola, mundo!' and 'Welcome to your new app.' Below this, there's a survey prompt: 'How is Blazor working for you? Please take our [brief survey](#) and tell us what you think.' On the left, there's a sidebar with three items: 'Home' (selected), 'Counter', and 'Fetch data'.

Este método es muy delicado porque si el día de mañana quisiéramos sacar el ! el key ya no sería el mismo y para que se siguiera traduciendo habría que corregir todos los keys de todos los idiomas (que pueden ser muchos).

Por tal motivo, es preferible utilizar la segunda técnica que involucra usar llaves y valores en todos los idiomas. Incluso, se pueden agregar una estructura que nos permita indicar el componente (por si necesitáramos distintas traducciones en diferentes componentes).

Por ejemplo:

Index.razor

```
@page "/"
@inject IStringLocalizer<Resource> localizer

<PageTitle>Index</PageTitle>

<h1>@localizer["Greeting"]</h1>

@localizer["index.welcome"]

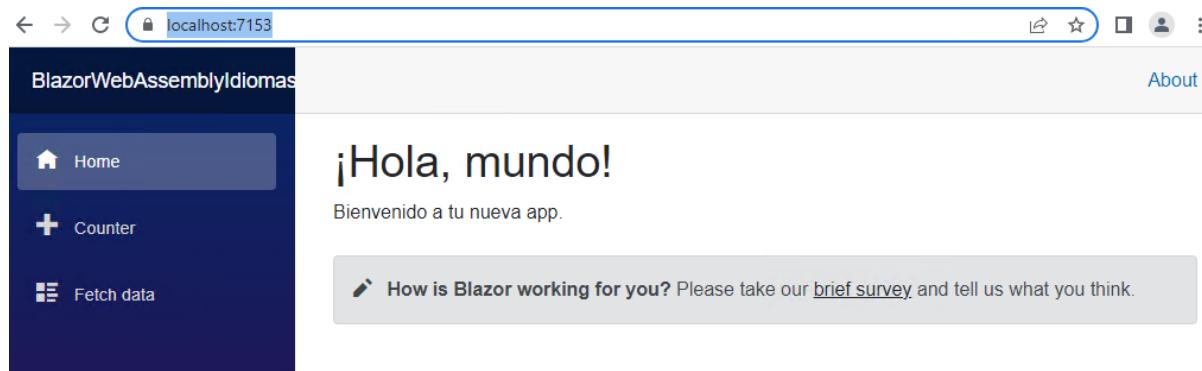
<SurveyPrompt Title="How is Blazor working for you?" />
```

Resource.resx

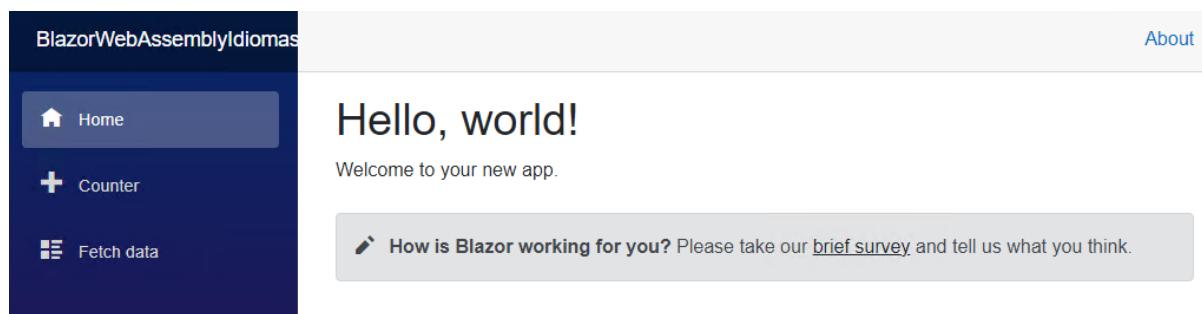
Name	Value
Greeting	Hello, world!
index.welcome	Welcome to your new app.

Resource.es.resx

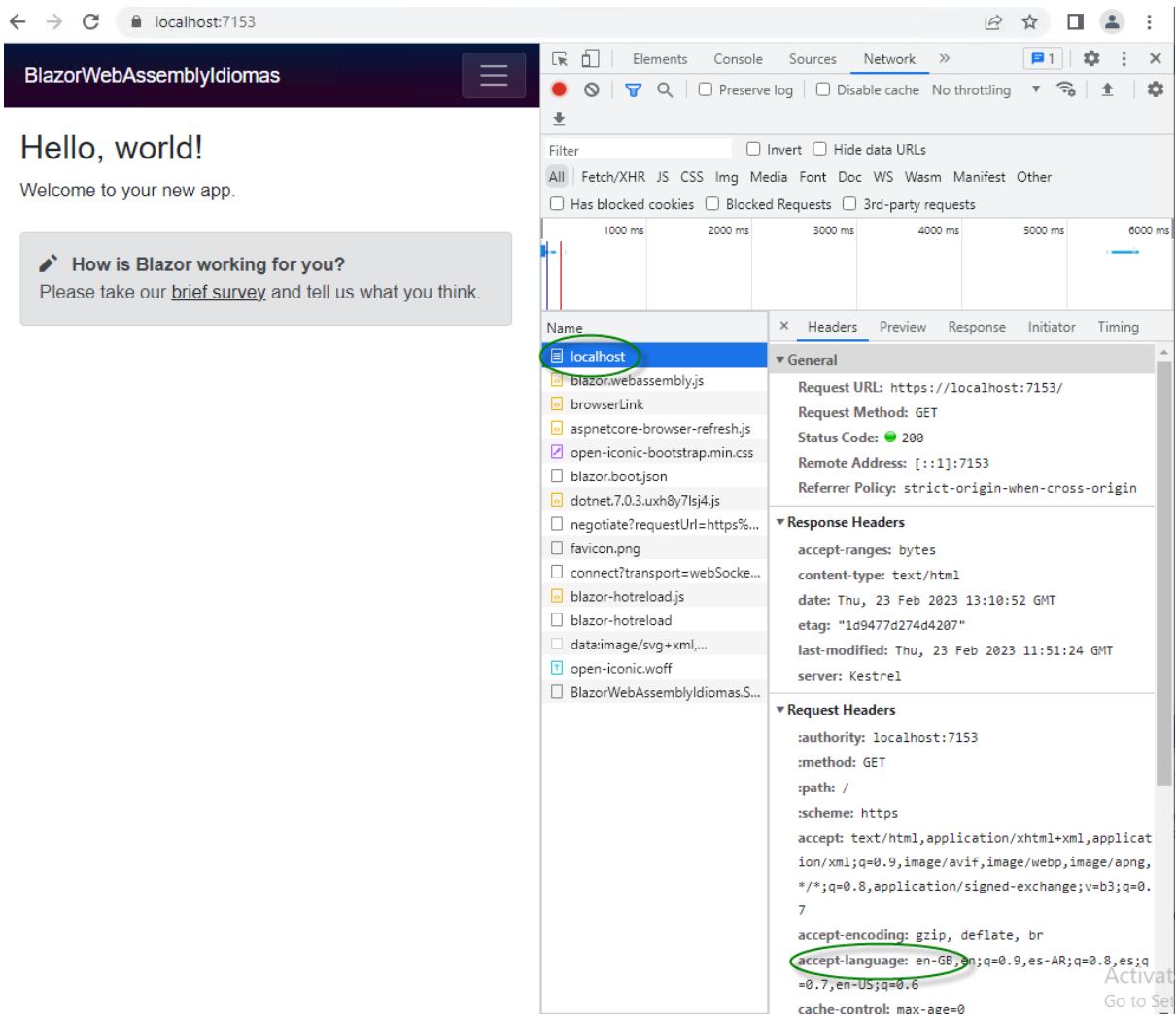
Name	Value
Greeting	¡Hola, mundo!
index.welcome	Bienvenido a tu nueva app.



Y si volvemos la configuración al inglés y refrescamos la página:



Recordamos que tenemos 3 proveedores de cultura. Uno de ellos es la cabecera HTTP. ¿Cómo se puede validar que estamos utilizando las cabecera **Accept-language**? Mirando los **Request Headers** de **localhost**:



The screenshot shows the Network tab of the Chrome DevTools Network panel. A request for 'localhost' is selected, highlighted with a green oval. The 'Headers' section shows the following headers:

Name	Value
:authority	localhost:7153
:method	GET
:path	/
:scheme	https
accept	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
accept-encoding	gzip, deflate, br
accept-language	en-GB,en;q=0.9,es-AR;q=0.8,es;q=0.7,en-US;q=0.6
cache-control	max-age=0

Si volvemos al español y refrescamos la página:

The screenshot shows a browser window for 'BlazorWebAssemblyIdiomas'. The main content area displays '¡Hola, mundo!' and a message encouraging users to take a survey. The browser's developer tools Network tab is open, showing a list of requests. The first request, 'localhost', is highlighted with a green circle. The 'Headers' section of the tool shows the following details:

```

Request URL: https://localhost:7153/
Request Method: GET
Status Code: 200
Remote Address: [::1]:7153
Referrer Policy: strict-origin-when-cross-origin
    
```

The 'Response Headers' section includes:

```

accept-ranges: bytes
content-type: text/html
date: Thu, 23 Feb 2023 13:12:26 GMT
etag: "1d9477d274d4207"
last-modified: Thu, 23 Feb 2023 11:51:24 GMT
server: Kestrel
    
```

The 'Request Headers' section includes:

```

:authority: localhost:7153
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
accept-encoding: gzip, deflate, br
accept-language: es-AR,en;q=0.9,en-GB;q=0.8,en-US;q=0.7,en;q=0.6
cache-control: max-age=0
    
```

Esto quiere decir que Chrome está mandando el idioma preferido y nuestra app utiliza esa info para mostrar los textos en el lenguaje deseado.

Cambio manual

Lo primero que haremos es crear un componente con un dropdown que permitirá elegir la cultura que desea el usuario. Crearemos el componente **CultureSelector.razor** en la carpeta **Shared**.

La clase **CultureInfo** sirve para encapsular la info de una cultura.

Creamos una variable culture que la usaremos en el **bind**. En su **get**, tomamos la cultura actual. En su **set**, validamos si es distinto a la actual para aplicar los cambios necesarios.

Usaremos **LocalStorage** para guardar la cultura preferida del usuario. Agregamos el siguiente código en index.html:

index.html

```
<!DOCTYPE html>
<html lang="en">

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

<body>
    <div id="app">
        <svg class="loading-progress">
            <circle r="40%" cx="50%" cy="50%" />
            <circle r="40%" cx="50%" cy="50%" />
        </svg>
        <div class="loading-progress-text"></div>
    </div>

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

    <script>
        window.culture = {
            get: () => window.localStorage['culture'],
            set: (value) => window.localStorage['culture'] = value
        };
    </script>
</body>

</html>
```

Esto es para obtener el valor actual y guardarlo en el LS y poder obtenerlo de ahí cuando sea necesario.

Como estamos dentro de una propiedad (set) utilizaremos programación síncrona. Como se trata de un componente nuevo, sólo resaltaremos las partes comentadas (explicadas) en los párrafos anteriores.

CultureSelector.razo,

```
@inject IJSRuntime js
@inject NavigationManager navManager
@using System.Globalization

<strong>Culture</strong>
<select @bind="culture">
    @foreach(var item in cultures) {
        <option value="@item">@item.DisplayName</option>
    }
</select>

@code {
    CultureInfo[] cultures = new[] {
        new CultureInfo("en-US"),
        new CultureInfo("es-AR"),
        new CultureInfo("es"),
        new CultureInfo("fr-FR")
    };

    CultureInfo culture {
        get => CultureInfo.CurrentCulture;
        set {
            if (CultureInfo.CurrentCulture != value) {
                var JsInProcessRuntime = (IJSInProcessRuntime)js; //Casteo para progr. síncrona
                JsInProcessRuntime.InvokeVoid("culture.set", value.Name); //Cultura a guardar. Por ej: en-US

                navManager.NavigateTo(navManager.Uri, forceLoad: true);
            }
        }
    }
}
```

Lo que necesitamos, ahora, es que nuestra app tome el valor de culture y lo aplique. Esto lo hacemos en [Program.cs](#).

Program.cs

```
using BlazorWebAssemblyIdiomas.Client;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Microsoft.JSInterop;
using System.Globalization;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
```

```
builder.RootComponents.Add<HeadOutlet>("head::after");

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

var host = builder.Build(); //Necesitamos una ref de IJSRuntime
var js = host.Services.GetRequiredService<IJSRuntime>();
var culture = await js.InvokeAsync<string>("culture.get");
var thisCulture = new CultureInfo("en-US");

if(culture != null)
    thisCulture = new CultureInfo(culture);

CultureInfo.DefaultThreadCurrentCulture = thisCulture;
CultureInfo.DefaultThreadCurrentUICulture = thisCulture;

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

Ahora tenemos que hacer una configuración en el **csproj** del proyecto **Client** para permitir la modificación anterior. Para ello, hacemos click-derecho sobre el proyecto **Client** y elegimos **Edit Project File**.

```
BlazorWebAssemblyIdiomas.Client.csproj
<Project Sdk="Microsoft.NET.Sdk.BlazorWebAssembly">

<PropertyGroup>
    <TargetFramework>net7.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>
    <BlazorWebAssemblyLoadAllGlobalizationData>true</BlazorWebAssemblyLoadAllGlobalizationData>
</PropertyGroup>

<ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly" Version="7.0.3" />
    <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.DevServer"
Version="7.0.3" PrivateAssets="all" />
    <PackageReference Include="Microsoft.Extensions.Localization" Version="7.0.3" />
</ItemGroup>

<ItemGroup>
    <ProjectReference Include=".\\Shared\\BlazorWebAssemblyIdiomas.Shared.csproj" />
</ItemGroup>

<ItemGroup>
    <Compile Update="Resources\Resource.Designer.cs">
        <DesignTime>True</DesignTime>
        <AutoGen>True</AutoGen>
    </Compile>
</ItemGroup>
```

```
<DependentUpon>Resource.resx</DependentUpon>
</Compile>
</ItemGroup>

<ItemGroup>
  <EmbeddedResource Update="Resources\Resource.resx">
    <Generator>PublicResXFileCodeGenerator</Generator>
    <LastGenOutput>Resource.Designer.cs</LastGenOutput>
  </EmbeddedResource>
</ItemGroup>

</Project>
```

La idea de esto es que como estamos realizando una modificación de la cultura en nuestro archivo **Program.cs**, necesitamos decirle a Blazor que permita realizar esta acción, porque por defecto esta acción no se permite y por tanto esto nos daría error si no realizamos la configuración en el archivo **csproj**.

Finalmente, ponemos el selector en el componente **MainLaout.razor**.

Logout.razor

```
@inherits LayoutComponentBase

<div class="page">
  <div class="sidebar">
    <NavMenu />
  </div>

  <main>
    <div class="top-row px-4">
      <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
      <CultureSelector />
    </div>

    <article class="content px-4">
      @Body
    </article>
  </main>
</div>
```

Al correrlo, vemos que la app se muestra en inglés porque es el idioma configurado por defecto a pesar de que Chrome le está indicando que el preferido es el español:

The screenshot shows a browser window with the URL `localhost:7153`. On the left, there's a sidebar with links for 'Home', 'Counter', and 'Fetch data'. The main content area displays the text 'Hello, world!' and a survey prompt: 'How is Blazor working for you? Please take our [brief survey](#) and tell us what you think.' Above the content, a 'Culture' dropdown menu is open, showing 'en (US)' with a green oval around it. To the right of the browser is the F12 developer tools Network tab. The 'Request Headers' section shows the following:

```

:authority: localhost:7153
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
accept-encoding: gzip, deflate, br
accept-language: es-AR,es;q=0.9,en-GB;q=0.8,en-US;q=0.7,en;q=0.
    
```

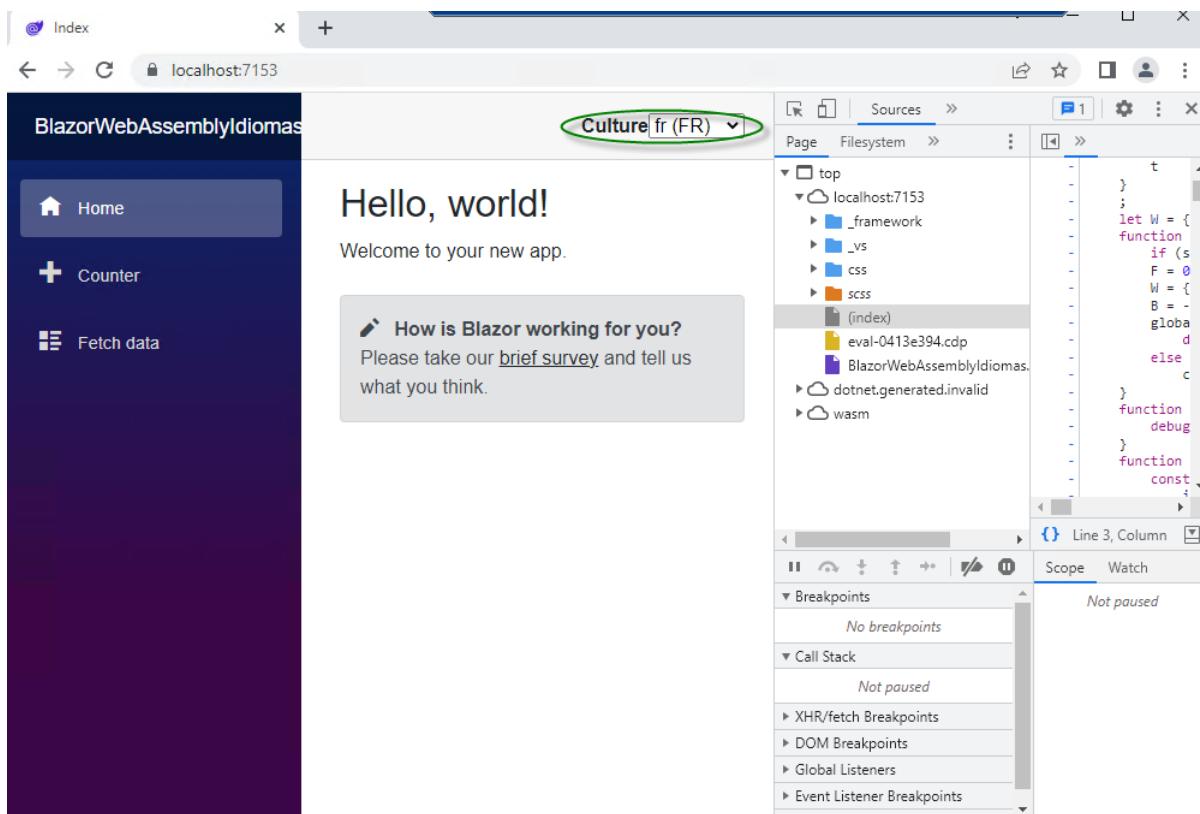
Al cambiar el selector a español, se refresca la página mostrándola en español:

The screenshot shows a browser window with the URL `localhost:7153`. The sidebar and content area are identical to the previous screenshot. However, the 'Culture' dropdown now shows 'es (AR)' with a green oval around it. The main content area displays the text '¡Hola, mundo!' and the same survey prompt. The F12 developer tools Network tab shows the following request headers:

```

:authority: localhost:7153
:method: GET
:path: /
:scheme: https
accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
accept-encoding: gzip, deflate, br
accept-language: es-AR,es;q=0.9,en-GB;q=0.8,en-US;q=0.7,en;q=0.
    
```

Al elegir, francés, en cambio se muestra en inglés porque no tenemos un recurso para ese idioma y se utilizan los valores por defecto (inglés):



Formato de fecha y números

Agregaremos una fecha y un monto formateado como **Currency** en el componente **Index.razor**.

```
Index.razor
@page "/"
@inject IStringLocalizer<Resource> localizer

<PageTitle>Index</PageTitle>

<h1>@localizer["Greeting"]</h1>

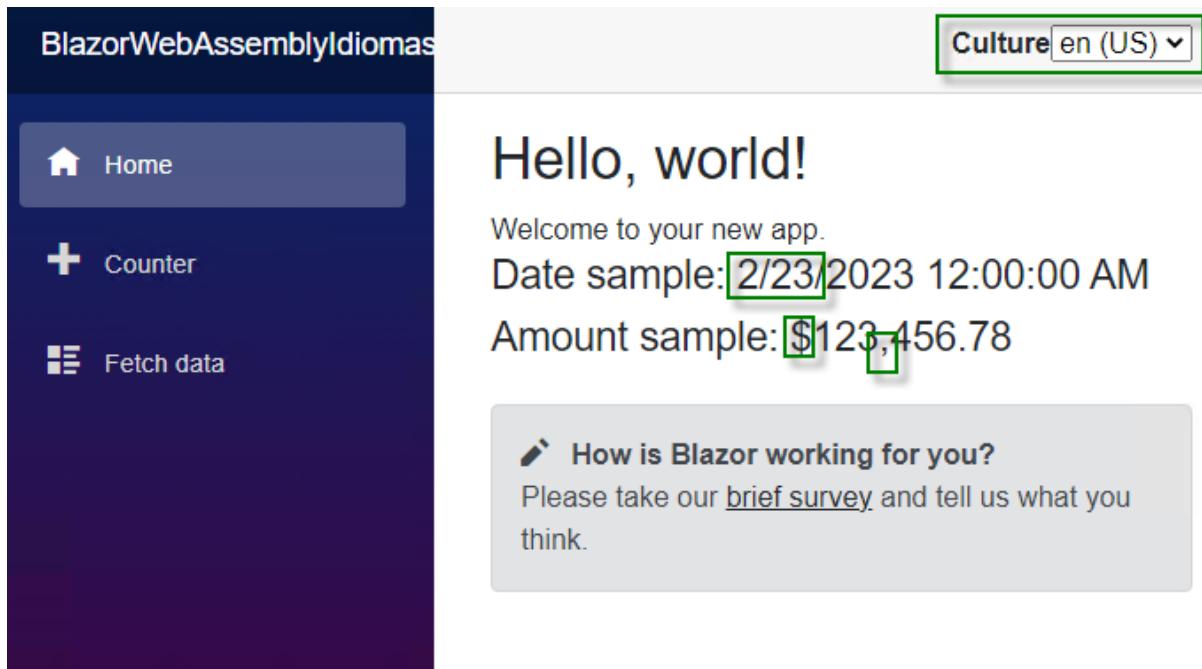
@localizer["index.welcome"]

<h4>Date sample: @DateTime.Today</h4>
<h4>Amount sample: @price.ToString("C")</h4>

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

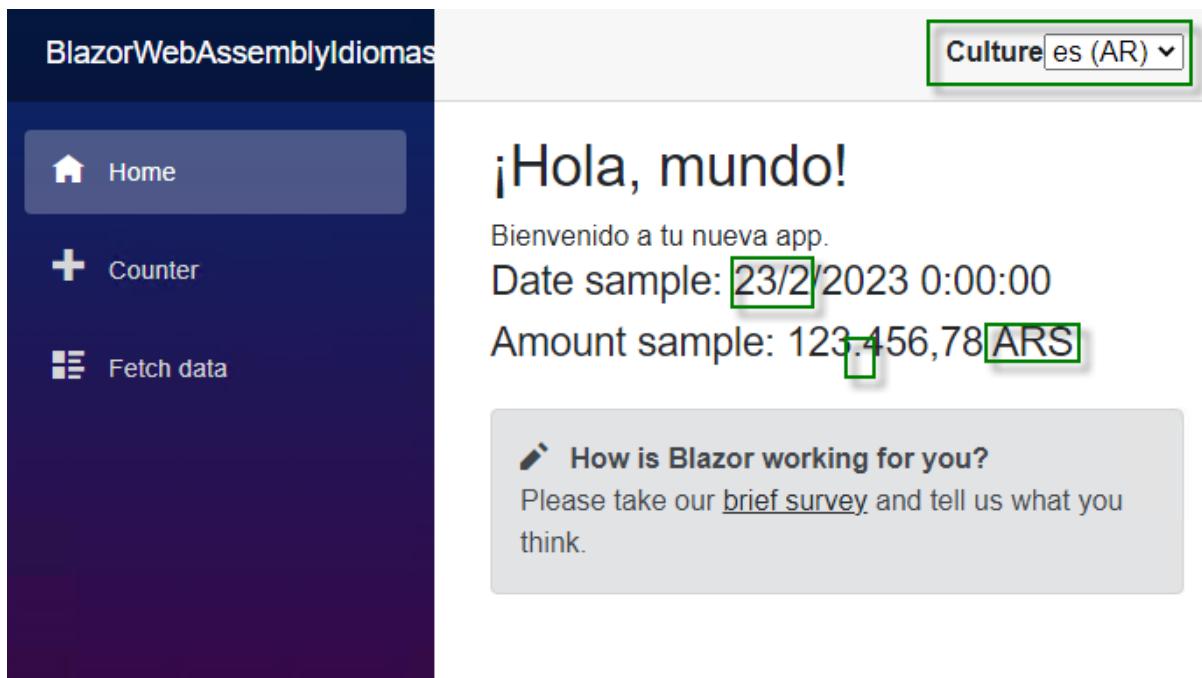
@code {
    decimal price = 123456.78m;
}
```

Corremos la app y vemos que cuando está en **en-US** las fechas se escriben como MM/DD/YYYY, el separador de miles es la coma, el separador decimal es el punto y el símbolo de moneda es el \$.



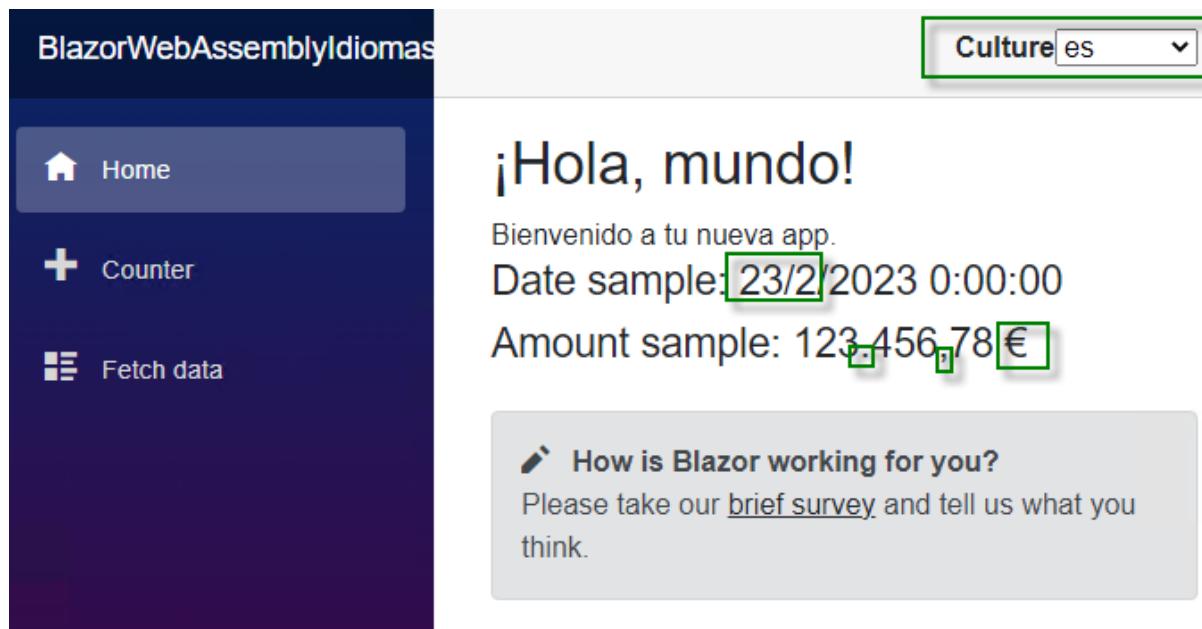
The screenshot shows a Blazor application interface. At the top left is the title "BlazorWebAssemblyIdiomas". On the right is a "Culture" dropdown menu set to "en (US)". The left sidebar has three items: "Home" (selected), "Counter", and "Fetch data". The main content area displays the text "Hello, world!", followed by "Welcome to your new app.", a date sample "Date sample: 2/23/2023 12:00:00 AM", and an amount sample "Amount sample: \$123,456.78". A survey prompt at the bottom right says "How is Blazor working for you? Please take our [brief survey](#) and tell us what you think." The date and amount values are highlighted with green boxes.

Y cuando cambiamos a **es-AR**, las fechas se escriben como DD/MM/YYYY, el separador de miles es el punto, el separador decimal es la coma y el símbolo de moneda es el ARS.



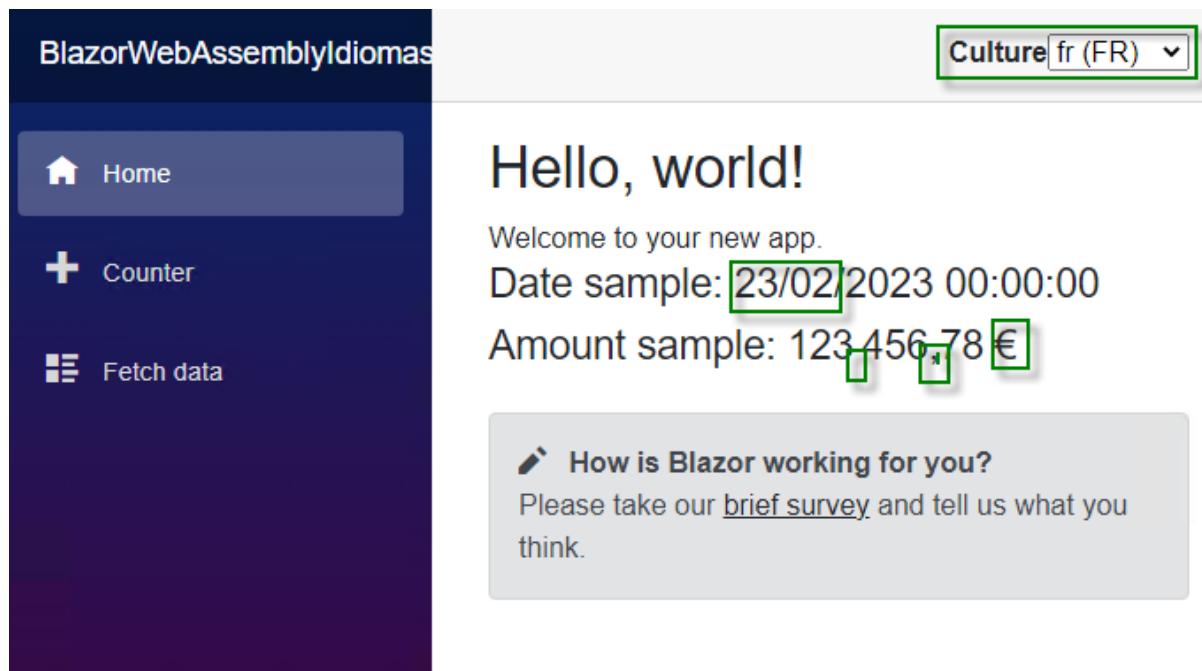
The screenshot shows the same Blazor application interface as above, but with the culture set to "es (AR)". The main content area now displays "¡Hola, mundo!", followed by "Bienvenido a tu nueva app.", a date sample "Date sample: 23/2/2023 0:00:00", and an amount sample "Amount sample: 123.456,78 ARS". The date and amount values are highlighted with green boxes. The survey prompt at the bottom right remains the same.

Y para **es**, la cultura por defecto es la de España. Por lo tanto, las fechas se escriben como DD/MM/YYYY, el separador de miles es el punto, el separador decimal es la coma y el símbolo de moneda es el €.



The screenshot shows a Blazor application interface. On the left is a sidebar with three items: "Home" (selected), "Counter", and "Fetch data". On the right, the main content area has a header "BlazorWebAssemblyIdiomas" and a dropdown menu showing "Culture es". Below the header, the text "¡Hola, mundo!" is displayed. A message below it says "Bienvenido a tu nueva app." followed by "Date sample: 23/2/2023 0:00:00" and "Amount sample: 123.456,78 €". At the bottom, there's a survey prompt: "How is Blazor working for you? Please take our [brief survey](#) and tell us what you think."

Y para **fr-FR**, las fechas se escriben como DD/MM/YYYY, el separador de miles es el espacio, el separador decimal es la coma y el símbolo de moneda es el €.



The screenshot shows a Blazor application interface. On the left is a sidebar with three items: "Home" (selected), "Counter", and "Fetch data". On the right, the main content area has a header "BlazorWebAssemblyIdiomas" and a dropdown menu showing "Culture fr (FR)". Below the header, the text "Hello, world!" is displayed. A message below it says "Welcome to your new app." followed by "Date sample: 23/02/2023 00:00:00" and "Amount sample: 123 456,78 €". At the bottom, there's a survey prompt: "How is Blazor working for you? Please take our [brief survey](#) and tell us what you think."

Sin cambios en cultura

Se puede conseguir que el cambio de una cultura sólo traduzca los textos pero no afecte al formato de números y fechas. Para esto, usamos **Culture** y **CultureUI**.

Como setaremos una cultura por defecto necesitamos cambiar nuestro selector para que use CultureUI.

```
CultureSelector.razor
@inject IJSRuntime js
@inject NavigationManager navManager
@using System.Globalization

<strong>Culture</strong>
<select @bind="culture">
    @foreach(var item in cultures) {
        <option value="@item">@item.DisplayName</option>
    }
</select>

@code {
    CultureInfo[] cultures = new[] {
        new CultureInfo("en-US"),
        new CultureInfo("es-AR"),
        new CultureInfo("es"),
        new CultureInfo("fr-FR")
    };

    CultureInfo culture {
        get => CultureInfo.CurrentCulture;
        set {
            if(CultureInfo.CurrentCulture != value) {
                var JsInProcessRuntime = (IJSInProcessRuntime)js; //Casteo para progr. síncrona
                JsInProcessRuntime.InvokeVoid("culture.set", value.Name); //Cultura a guardar. Por ej: en-US

                navManager.NavigateTo(navManager.Uri, forceLoad: true);
            }
        }
    }
}
```

Ahora sí podemos forzar la cultura para que no se afecten los números o fechas aún cambiando de idioma:

```
MainLayout.razor
@inherits LayoutComponentBase
@using System.Globalization

<div class="page">
    <div class="sidebar">
```

```

<NavMenu />
</div>

<main>
    <div class="top-row px-4">
        <CultureSelector />
    </div>

    <article class="content px-4">
        @Body
    </article>
</main>
</div>

@code {
    protected override void OnInitialized()
    {
        CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("es-AR");
    }
}

```

Veamos en las siguientes imágenes que el cambio de selector sólo afecta al idioma pero los formatos de fechas/números se mantienen según el defecto (es-AS).

The figure consists of four separate screenshots of a Blazor application interface, each demonstrating a different culture setting:

- Culture: es (AR)**: The interface is in Spanish (Argentina). The greeting is "¡Hola, mundo!", the date is "Date sample: 23/2/2023 0:00:00", and the amount is "Amount sample: 123.456,78 ARS". The survey prompt asks about Blazor usage.
- Culture: en (US)**: The interface is in English (United States). The greeting is "Hello, world!", the date is "Date sample: 23/2/2023 0:00:00", and the amount is "Amount sample: 123.456,78 ARS". The survey prompt asks about Blazor usage.
- Culture: es**: The interface is in Spanish (Argentina). The greeting is "¡Hola, mundo!", the date is "Date sample: 23/2/2023 0:00:00", and the amount is "Amount sample: 123.456,78 ARS". The survey prompt asks about Blazor usage.
- Culture: fr (FR)**: The interface is in French (France). The greeting is "Hello, world!", the date is "Date sample: 23/2/2023 0:00:00", and the amount is "Amount sample: 123.456,78 ARS". The survey prompt asks about Blazor usage.

También se puede forzar una cultura independientemente de lo que se haya seleccionado:

Index.razor

```
@page "/"
@inject IStringLocalizer<Resource> localizer

<PageTitle>Index</PageTitle>

<h1>@localizer["Greeting"]</h1>

@localizer["index.welcome"]

<h4>Date sample: @DateTime.Today.ToString("yyyy/MM/dd")</h4>
<h4>Amount sample: @price.ToString("C")</h4>

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

@code {
    decimal price = 123456.78m;
}
```


Traduciendo errores

Creamos una nueva clase **Person.cs** en el proyecto **Shared**.

Person.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorWebAssemblyIdiomas.Shared {
    public class Person {
        [Required]
        public string Name { get; set; } = null!;
    }
}
```

Agregamos el **using** en **_Imports.razor**.

_Imports.razor

```
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorWebAssemblyIdiomas.Client
@using BlazorWebAssemblyIdiomas.Client.Shared
@using Microsoft.Extensions.Localization
@using BlazorWebAssemblyIdiomas.Client.Resources
@using BlazorWebAssemblyIdiomas.Shared
```

Agregamos un item en NavMenu para poder acceder al componente que crearemos a continuación:

NavMenu.razor

```
<div class="top-row ps-3 navbar navbar-dark">
<div class="container-fluid">
    <a class="navbar-brand" href="">BlazorWebAssemblyIdiomas</a>
    <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
        <span class="navbar-toggler-icon"></span>
    </button>
</div>
</div>
```

```

<div class="@NavControllerCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="oi oi-plus" aria-hidden="true"></span> Counter
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="form">
                <span class="oi oi-plus" aria-hidden="true"></span> Form
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="fetchdata">
                <span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
            </NavLink>
        </div>
    </nav>
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}

```

Agregamos entradas en los 2 archivos para el Nombre y el título del componente:

Resource.resx	
Name	Value
Greeting	Hello, world!
index.welcome	Welcome to your new app.
Form.Name	Name
Form.Form	Form

Resource.es.resx	
Name	Value
Greeting	¡Hola, mundo!
index.welcome	Bienvenido a tu nueva app.
Form.Name	Nombre
Form.Form	Formulario

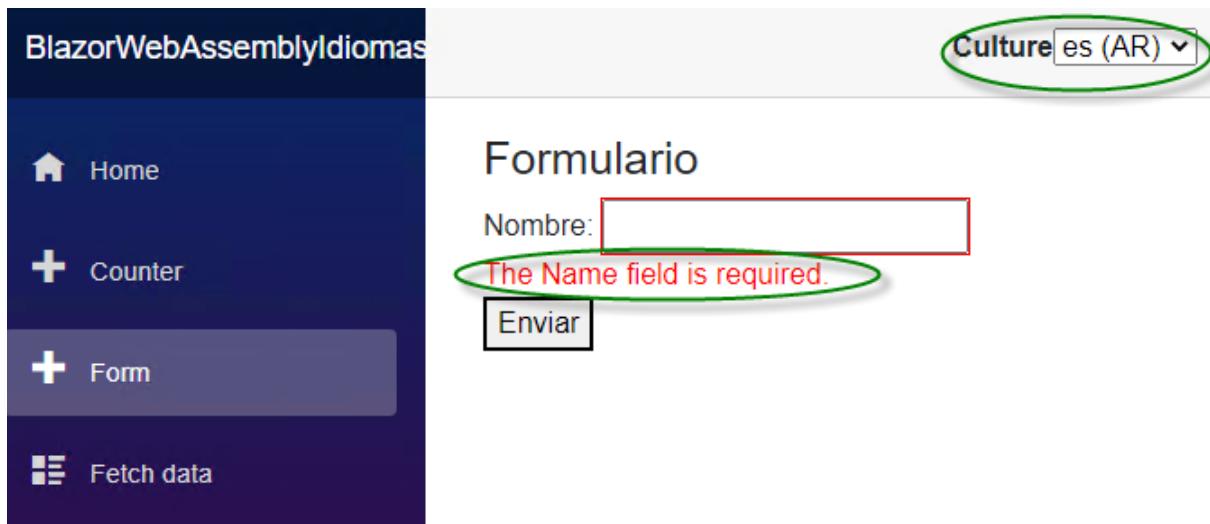
Creamos el componente **Form.razor** en la carpeta **Pages** del proyecto **Client**:

```
Form.razor
@page "/form"
@inject IStringLocalizer<Resource> localizer

<h3>@localizer["Form.Form"]</h3>

<EditForm Model="person">
    <DataAnnotationsValidator />
    <div>
        @localizer["Form.Name"]:
        <InputText @bind-Value="person.Name" />
        <ValidationMessage For="@(() => person.Name)"></ValidationMessage>
    </div>
    <button>@localizer["Form.Send"]</button>
</EditForm>
@code {
    Person person = new Person();
}
```

Abrimos Form y presionamos Send. Si bien estamos viendo la interfaz en español, es mensajes es en inglés:



Esto es porque no le hemos dicho a nuestro atributo de Required que utilice un recurso. Sin embargo, recordemos que nuestra carpeta **Resources** está en el proyecto **Client**. Mientras que nuestro modelo de personas se encuentra en el proyecto **Shared**. La dependencia que existe entre **Client** y **Shared** es que la primera depende de la segunda, pero no al revés porque sería una referencia circular.

Movemos, entonces, la carpeta **Resources** desde el proyecto **Client** al proyecto **Shared**.

Para que todo siga funcionando, es necesario corregir el componente **_Imports.razor** para que tenga el namespace correcto:

Logout.razor

```
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorWebAssemblyIdiomas.Client
@using BlazorWebAssemblyIdiomas.Client.Shared
@using Microsoft.Extensions.Localization
@using BlazorWebAssemblyIdiomas.Shared.Resources
@using BlazorWebAssemblyIdiomas.Shared
```

Realizamos un rebuild para que re-genere todo lo necesario.

En el atributo **Required** podemos indicar **ErrorMessageResourceType** y **ErrorMessageResourceName**. Con estos 2 podemos indicar el recurso a utilizar y el mensaje del recurso a utilizar.

Agregamos los mensajes en los archivos de recursos:

Resource.resx	
Name	Value
Greeting	Hello, world!
index.welcome	Welcome to your new app.
Form.Name	Name
Form.Form	Form
required	This field is required

Resource.es.resx	
Name	Value
Greeting	¡Hola, mundo!
index.welcome	Bienvenido a tu nueva app.
Form.Name	Nombre
Form.Form	Formulario
required	Este campo es requerido.

Agregamos **ErrorMessageResourceType** y **ErrorMessageResourceName** en el atributo de **Required** para indicar cuál es el recurso que debe utilizar y cuál es el mensaje que debe tomar de éste:

```
Logout.razor
using BlazorWebAssemblyIdiomas.Shared.Resources;
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorWebAssemblyIdiomas.Shared {
    public class Person {
        [Required(ErrorMessageResourceType = typeof(Resource),
        ErrorMessageResourceName = nameof(Resource.required))]
        public string Name { get; set; } = null!;
    }
}
```

Como se ve en las imágenes anteriores, el mensaje de error se muestra en el mismo idioma de la interfaz. Con esto, estamos traduciendo los mensajes de error de DataAnnotations para nuestros formularios.

Internacionalización en Blazor-Server

Creamos una nueva app tipo **Blazor Server App** llamada **BlazorServerIdiomas**. Lo primero que tenemos que hacer es configurar la localización en nuestra app para tener acceso al **IStringLocalizer**:

Program.cs

```
using BlazorServerIdiomas.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddLocalization();
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if(!app.Environment.IsDevelopment()) {
    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.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Creamos la carpeta **Resources** y en ella los archivos **Resource.resx** y **Resource.es.resx**. Setteamos el **Access Modifier** el primer archivo como **Public**.

Resource.resx

Name	Value
Greeting	Hello, world!

Resource.es.resx	
Name	Value
Greeting	¡Hola, mundo!

Realizamos un build para asegurarnos de que se generen las clases de referencias necesarias para poder utilizar los archivos de recursos con el `IStringLocalizer`.

Agregamos los `using` en `_Imports.razor`:

```
Logout.razor
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.JSInterop
@using BlazorServerIdiomas
@using BlazorServerIdiomas.Shared
@using Microsoft.Extensions.Localization
@using BlazorServerIdiomas.Resources
```

A diferencia de **WebAssembly**, tenemos que declarar las culturas soportadas a nivel de la clase **Program.cs**. Es importante agregar culturas que sólo incluya el idioma porque, a veces, el navegador sólo envía este dato.

```
Program.cs
using BlazorServerIdiomas.Data;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Localization;
using System.Globalization;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddLocalization();
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();

var app = builder.Build();

var SupportedCultures = new[] {
```

```
new CultureInfo("en-US"),
new CultureInfo("es-US"),
new CultureInfo("es-AR"),
new CultureInfo("es"),
new CultureInfo("us")
};

//Usamos un middleWare llamado UseRequestLocalization
app.UseRequestLocalization(new RequestLocalizationOptions {
    SupportedCultures = SupportedCultures,
    SupportedUIT Cultures = SupportedCultures,
    DefaultRequestCulture = new RequestCulture("en-US")
});

// Configure the HTTP request pipeline.
if(!app.Environment.IsDevelopment()) {
    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.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

El siguiente paso es utilizar cookies para poder establecer el idioma de preferencia del usuario. Eso es muy importante debido a que estamos utilizando web sockets. Al utilizar cookies nos aseguramos de que siempre la aplicación de ASP.NET Core tenga la información del idioma de preferencia del usuario.

Esto lo hacemos con un archivo de code-behind que nos permitirá ejecutar código personalizado cuando el usuario entre a la app. En ese momento, estableceremos las cookies. Agregamos un clase `_Host.cshtml.cs` en la carpeta `Pages`. Le cambiamos el nombre a la clase a `HostModel` y la hacemos heredar de `PageModel`.

```
_Host.cshtml.cs
using Microsoft.AspNetCore.Mvc.RazorPages;

namespace BlazorServerIdiomas.Pages {
```

```
public class _HostModel: PageModel {  
}  
}
```

Vamos a utilizar el método **OnGet** para ejecutar una función personalizada cuando el usuario entre la aplicación.

```
_Hostcshtml.cs  
using Microsoft.AspNetCore.Localization;  
using Microsoft.AspNetCore.Mvc.RazorPages;  
using System.Globalization;  
  
namespace BlazorServerIdiomas.Pages {  
    public class _HostModel: PageModel {  
        public void OnGet() {  
            //Agregamos una cookie con la estructura necesaria.  
            HttpContext.Response.Cookies.Append(  
                CookieRequestCultureProvider.DefaultCookieName,  
                CookieRequestCultureProvider.MakeCookieValue(  
                    new RequestCulture(  
                        CultureInfo.CurrentCulture,  
                        CultureInfo.CurrentUICulture  
                    )  
                );  
            );  
        }  
    }  
}
```

Con eso estamos seteando una cookie cuando el usuario entra a la aplicación y ésta va a contener la información de la cultura del usuario y se va a enviar a través de nuestra conexión de web socket. Recordemos que **BlazorServer** nos comunicamos utilizando una conexión **SignalR** para tener una comunicación en tiempo real con nuestra aplicación que se está ejecutando en un servidor web.

Si corremos veríamos que se muestra en el idioma en el que está configurado el navegador. Aún si lo cambiáramos y resfrescáramos la página... el idioma no cambiaría.

Esto se debe a que la cookie se setea con los valores del usuario y luego se envía esta cookie al server para que se configure en el **OnGet** con esos valores sin tomar en cuenta lo que se encuentra en la cabecera **Accept-language**. Esto se puede resolver con selección manual de la cultura.

Selección manual

Comentaremos el código de OnGet porque no lo necesitaremos.

_Host.cshtml.cs

```
using Microsoft.AspNetCore.Localization;
using Microsoft.AspNetCore.Mvc.RazorPages;
using System.Globalization;

namespace BlazorServerIdiomas.Pages {
    public class _HostModel: PageModel {
        public void OnGet() {
            //Agregamos una cookie con la estructura necesaria.
            //HttpContext.Response.Cookies.Append(
            //    CookieRequestCultureProvider.DefaultCookieName,
            //    CookieRequestCultureProvider.MakeCookieValue(
            //        new RequestCulture(
            //            CultureInfo.CurrentCulture,
            //            CultureInfo.CurrentCulture
            //        )
            //    )
            //);
        }
    }
}
```

Crearemos un nuevo controlador que correrá cuando el usuario cambie la cultura de la app. Creamos la carpeta **Controllers** y en ella una clase **CultureController.cs** y la hacemos heredar de **Controller**.

Cuando invoquemos este controlador, vamos a setear la cookie con la cultura enviada.

CultureController.cs

```
using Microsoft.AspNetCore.Localization;
using Microsoft.AspNetCore.Mvc;

namespace BlazorServerIdiomas.Controllers {
    [Route("[controller]/[action]")]
    public class CultureController: Controller {
        public IActionResult SetCulture(string culture, string redirectURL) {
            if(culture != null) {
                HttpContext.Response.Cookies.Append(
                    CookieRequestCultureProvider.DefaultCookieName,
                    CookieRequestCultureProvider.MakeCookieValue(
                        new RequestCulture(culture)
                    );
            }
            return LocalRedirect(redirectURL);
        }
    }
}
```

Crearemos un componente llamado **CultureSelector** en la carpeta **Shared** y copiamos el código del proyecto de **WebAssembly** y haremos algunos cambios. El primero... es sacar las culturas que están hardcodeadas ya que en Program.cs ya habíamos configurado las culturas soportadas.

Para no repetir código, crearemos una carpeta **Helpers** y en ella la clase estática **Constants.cs**:

```
Constants.cs
using System.Globalization;

namespace BlazorServerIdiomas.Helpers {
    public static class Constants {
        public static CultureInfo[] SupportedCultures = new[] {
            new CultureInfo("en-US"),
            new CultureInfo("es-US"),
            new CultureInfo("es-AR"),
            new CultureInfo("es"),
            new CultureInfo("us")
        };
    }
}
```

Ya podemos sacar el hardcode en Program.cs:

```
Program.cs
using BlazorServerIdiomas.Data;
using BlazorServerIdiomas.Helpers;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Localization;
using System.Globalization;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddLocalization();
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();

var app = builder.Build();

//Usamos un middleWare llamado UseRequestLocalization
app.UseRequestLocalization(new RequestLocalizationOptions {
    SupportedCultures = Constants.SupportedCultures,
    SupportedUIT Cultures = Constants.SupportedCultures,
```

```
DefaultRequestCulture = new RequestCulture("en-US")
});

// Configure the HTTP request pipeline.
if(!app.Environment.IsDevelopment()) {
    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.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Y agregamos el **using** en **_Imports.razor**.

```
_Imports.razor
@using System.Net.Http
@using Microsoft.AspNetCore.Authorization
@using Microsoft.AspNetCore.Components.Authorization
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.JSInterop
@using BlazorServerIdiomas
@using BlazorServerIdiomas.Shared
@using Microsoft.Extensions.Localization
@using BlazorServerIdiomas.Resources
@using BlazorServerIdiomas.Helpers
```

Y realizamos el cambio en el componente CultureSelector para que utilice el controlador creado:

```
CultureSelector.razor
@inject IJSRuntime js
@inject NavigationManager navManager
@using System.Globalization
```

```

<strong>Culture</strong>
<select @bind="culture">
    @foreach(var item in cultures) {
        <option value="@item">@item.DisplayName</option>
    }
</select>

@code {
    CultureInfo[] cultures = Constants.SupportedCultures;

    CultureInfo culture {
        get => CultureInfo.CurrentCulture;
        set {
            if(CultureInfo.CurrentCulture != value) {
                var culture = value.Name;
                var uri = new Uri(navManager.Uri)
                    .GetComponents(UriComponents.PathAndQuery, UriFormat.Unescaped);
                var query =
                    $"?culture={Uri.EscapeDataString(culture)}&redirectURL={Uri.EscapeDataString(uri)}";

                navManager.NavigateTo("/culture/SetCulture" + query, forceLoad: true);
            }
        }
    }
}
    
```

Incorporamos el componente creado en el componente **MainLayout.razor**:

```

MainLayout.razor
@inherits LayoutComponentBase

<PageTitle>BlazorServerIdiomas</PageTitle>

<div class="page">
    <div class="sidebar">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <CultureSelector />
        </div>

        <article class="content px-4">
            @Body
        </article>
    </main>
</div>
    
```

Sólo falta permitir que los usuarios puedan acceder a los controladores de nuestra app de **BlazorServer** que por defecto no tiene esa configuración.

Program.cs

```
using BlazorServerIdiomas.Data;
using BlazorServerIdiomas.Helpers;
using Microsoft.AspNetCore.Components;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Localization;
using System.Globalization;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddLocalization();
// Add services to the container.
builder.Services.AddRazorPages();
builder.Services.AddServerSideBlazor();
builder.Services.AddSingleton<WeatherForecastService>();

var app = builder.Build();

//Usamos un middleWare llamado UseRequestLocalization
app.UseRequestLocalization(new RequestLocalizationOptions {
    SupportedCultures = Constants.SupportedCultures,
    SupportedUIT Cultures = Constants.SupportedCultures,
    DefaultRequestCulture = new RequestCulture("en-US")
});

// Configure the HTTP request pipeline.
if(app.Environment.IsDevelopment()) {
    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.MapControllers();
app.MapBlazorHub();
app.MapFallbackToPage("/_Host");

app.Run();
```

Curso: t.ly/42al

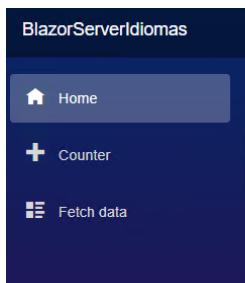
Docente: Felipe Gavilán (t.ly/la_0)

Resumió: Ing. Emiliano F. MARTÍN (<http://emilianomartin.com.ar/CV/index.php>)

Curso: t.ly/42al

Docente: Felipe Gavilán (t.ly/la_0)

Resumió: Ing. Emiliano F. MARTÍN (<http://emilianomartin.com.ar/CV/index.php>)



Culture English (United States) ▾

Hello, world!

Welcome to your new app.

✍ How is Blazor working for you?
Please take our [brief survey](#) and tell us what you think.

Culture español (Argentina) ▾

Hello, world!

Welcome to your new app.

✍ How is Blazor working for you?
Please take our [brief survey](#) and tell us what you think.

Resumen

-  Internacionalización es el proceso de diseñar nuestras aplicaciones para que se puedan adaptar a distintos idiomas y regiones
-  La globalización es la parte en la cual diseñamos estas aplicaciones internacionalizadas
-  La localización se refiere al proceso de adaptar una aplicación globalizada a una cultura específica
-  Blazor WebAssembly y Blazor Server
-  Nuestros usuarios pueden escoger manualmente el idioma
-  Archivos de recursos
-  Mensajes de error en distintos idiomas

Progressive Web Apps

Las aplicaciones web progresivas nos permiten utilizar APIs nativos de los navegadores para realizar aplicaciones web con una nueva dimensión de funcionalidades.

Las aplicaciones web progresivas (PWA) nos permiten crear aplicaciones web, las cuales tienen capacidades de instalación, trabajar en modo offline e incluso recibir notificaciones, aún cuando el usuario no se encuentra utilizando nuestra aplicación en ese momento.

Es decir, con las PWA podemos crear aplicaciones web que tienen algunas funcionalidades de las aplicaciones nativas.

Lo bueno de esta tecnología es que es progresiva. Eso significa que si el navegador del usuario no soporta una característica X, como las notificaciones. De todos modos, la aplicación puede funcionar.

Así tendremos un conjunto de funcionalidad básica de la aplicación que siempre debe estar presente.

Luego, dependiendo del deseo del usuario y las capacidades del navegador, puede optar por funcionalidad extra. Tal como dijimos, una de estas características es el modo offline. Con este modo, el usuario puede usar nuestra aplicación sin necesidad de conexión a Internet. Esto funciona gracias a que guardamos en caché los recursos de la página. De esta manera, si no hay Internet, el usuario puede interactuar con nuestra página valiéndose de la caché del navegador. Luego, cuando haya conexión, el usuario puede continuar usando la aplicación.

Para guardar los recursos del sitio web en caché utilizamos un ServiceWorker.

Un ServiceWorker es un archivo especial de JavaScript que puede interceptar la comunicación entre tu aplicación y un servidor web. Es decir, actúa como un proxy.

En este archivo de JavaScript podemos interceptar eventos, por ejemplo, podemos disparar un código cada vez que la aplicación web haga una petición HTTP hacia el servidor.

También podemos ejecutar un código cuando un servidor intenta hacer un push hacia nuestra aplicación. Con este push, nuestro ServiceWorker puede mostrar una notificación al usuario.

Las PWA también son instalables. Esto quiere decir que el usuario va a poder acceder a nuestro sitio web en una ventana especial sin necesidad de utilizar el navegador.

Podemos instalar las PWA tanto en escritorios, ya sea de Windows o Mac, como es como dispositivos móviles como Android, y iOS.

Podemos usar PWA cuando necesitamos hacer una aplicación, la cual no necesita capacidades nativas, pero que opcionalmente podremos beneficiarnos de algunas de esas capacidades. Y claro está, también el hecho de que si tenemos una PWA vamos a poder compartir código entre nuestra aplicación web y la aplicación que los usuarios van a instalar en sus celulares.

Como hemos mencionado, esas capacidades incluyen instalación, modo offline, notificaciones por eventos push, entre otros. Sin embargo, PWA no contiene todas las funcionalidades. Por ejemplo, alguna de sus limitaciones son limitaciones de ejecución en segundo plano, I/O fencing, que es cuando tú ejecutas código según la ubicación del usuario.

Así, por ejemplo, si tu quieres ejecutar un código cuando el usuario entre a un edificio en específico, tú puedes hacerlo con I/O fencing, pero PWA no soporta esto.

También ser notificados de eventos del sistema operativo, por ejemplo, en el caso de un celular algunas aplicaciones pueden saber cuándo entra una llamada al celular.

Esto no es algo que podemos hacer con los PWA. Es decir, este tipo de eventos de sistema operativo no podemos ni interceptarlos, ni siquiera están enterados de su existencia.

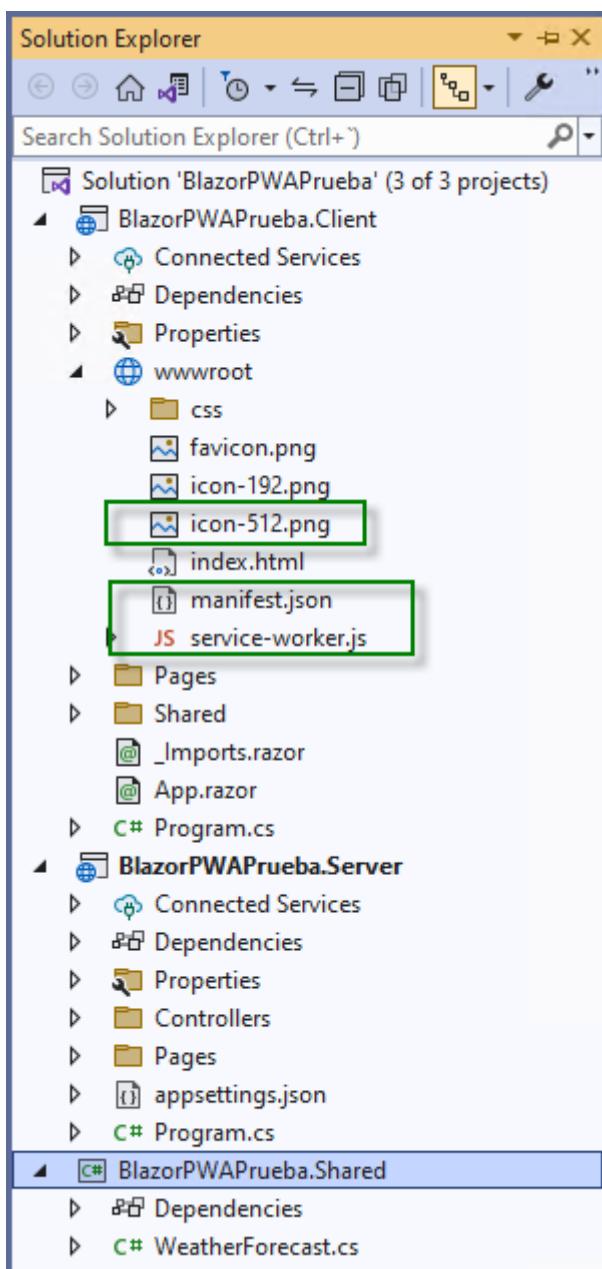
Diferencia entre PWA y una aplicación híbrida

Las aplicaciones híbridas son parecidas a las de PWA, pues ambas tienen código de HTML, CSS y JavaScript.

La diferencia es que mientras las PWA son puro HTML, CSS y JavaScript, las aplicaciones híbridas también tienen código nativo. Es decir, podemos ver una aplicación híbrida como una aplicación web envuelta dentro de una aplicación nativa.

Por tanto, como las PWA no utilizamos tecnología nativa, no se consideran híbridas, sino aplicaciones web que aprovechan funcionalidades especiales para tener algunas de las características de aplicaciones nativas.

Iniciamos una nueva. Buscamos Blazor y elegimos **Blazor WebAssembly App**. Tildamos **ASP.NET Core Hosted** y **Progressive Web Application**, que me permitirá tener un web api acompañando a mi app de Blazor (en esta accederemos a una BD).



Las mayores diferencias se dan en la carpeta **wwwroot** del proyecto **Client**. Por ejemplo, el archivo **manifest.json** que es el manifiesto de nuestra app en el que está el nombre de la misma, la web, el inicio y algunos temas de diseño que son importantes para cuando el usuario instale la PWA en su equipo como, por ejemplo, el ícono de la app.

manifest.json

```
{
    "name": "BlazorPWAPrueba",
    "short_name": "BlazorPWAPrueba",
    "start_url": "./",
    "display": "standalone",
    "background_color": "#ffffff",
    "theme_color": "#03173d",
    "prefer_related_applications": false,
    "icons": [
        {
            "src": "icon-512.png",
            "type": "image/png",
            "sizes": "512x512"
        },
        {
            "src": "icon-192.png",
            "type": "image/png",
            "sizes": "192x192"
        }
    ]
}
```

También está el ServiceWorker. Es el archivo que hace de proxy e intercepta determinados eventos. Por ejemplo cuando nuestra app intenta enviar una petición HTTP hacia el servidor web. Esta petición puede ser interceptada y hacer con esa operación lo que queramos.

Por ejemplo, si el usuario no tuviera conexión, podemos darle respuesta desde el caché y, así, implementaríamos el modo offline.

En el index.html tenemos el manifest, los íconos y el serviceworker:

index.html

```
<!DOCTYPE html>
<html lang="en">
```

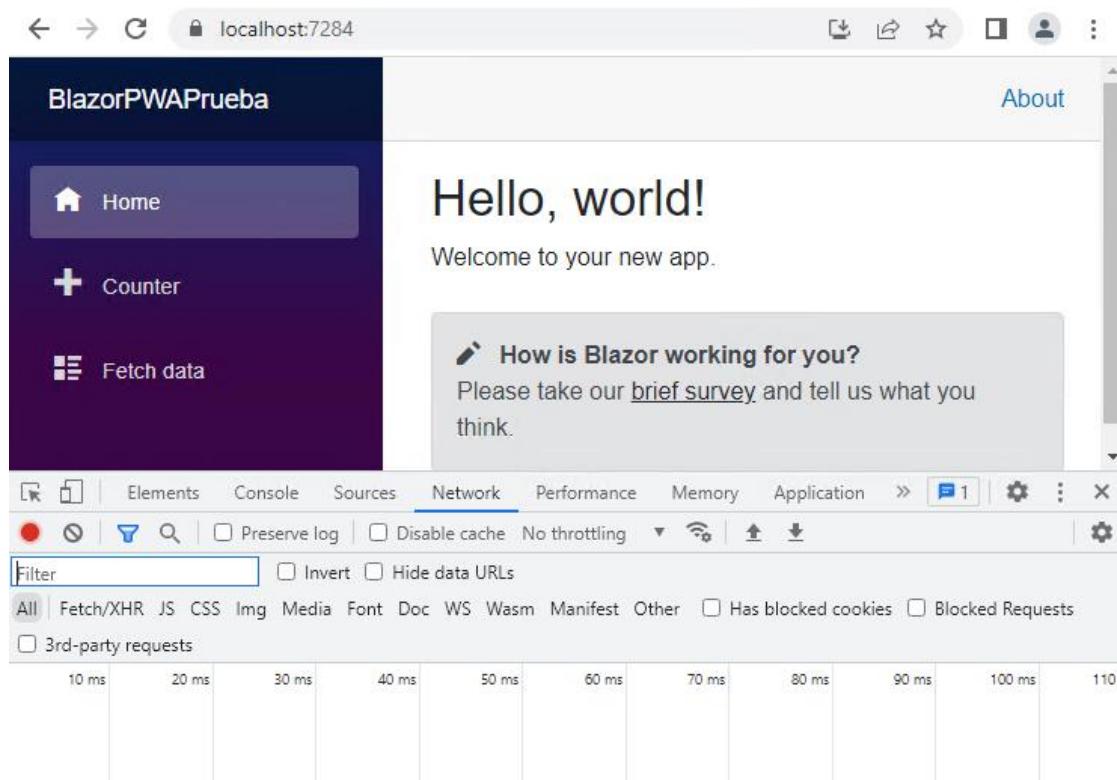
```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
  <title>BlazorPWAPrueba</title>
  <base href="/" />
  <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
  <link href="css/app.css" rel="stylesheet" />
  <link rel="icon" type="image/png" href="favicon.png" />
  <link href="BlazorPWAPrueba.Client.styles.css" rel="stylesheet" />
  <link href="manifest.json" rel="manifest" />
  <link rel="apple-touch-icon" sizes="512x512" href="icon-512.png" />
  <link rel="apple-touch-icon" sizes="192x192" href="icon-192.png" />
</head>

<body>
  <div id="app">
    <svg class="loading-progress">
      <circle r="40%" cx="50%" cy="50%" />
      <circle r="40%" cx="50%" cy="50%" />
    </svg>
    <div class="loading-progress-text"></div>
  </div>

  <div id="blazor-error-ui">
    An unhandled error has occurred.
    <a href="" class="reload">Reload</a>
    <a class="dismiss">X</a>
  </div>
  <script src="_framework/blazor.webassembly.js"></script>
  <script>navigator.serviceWorker.register('service-worker.js');</script>
</body>

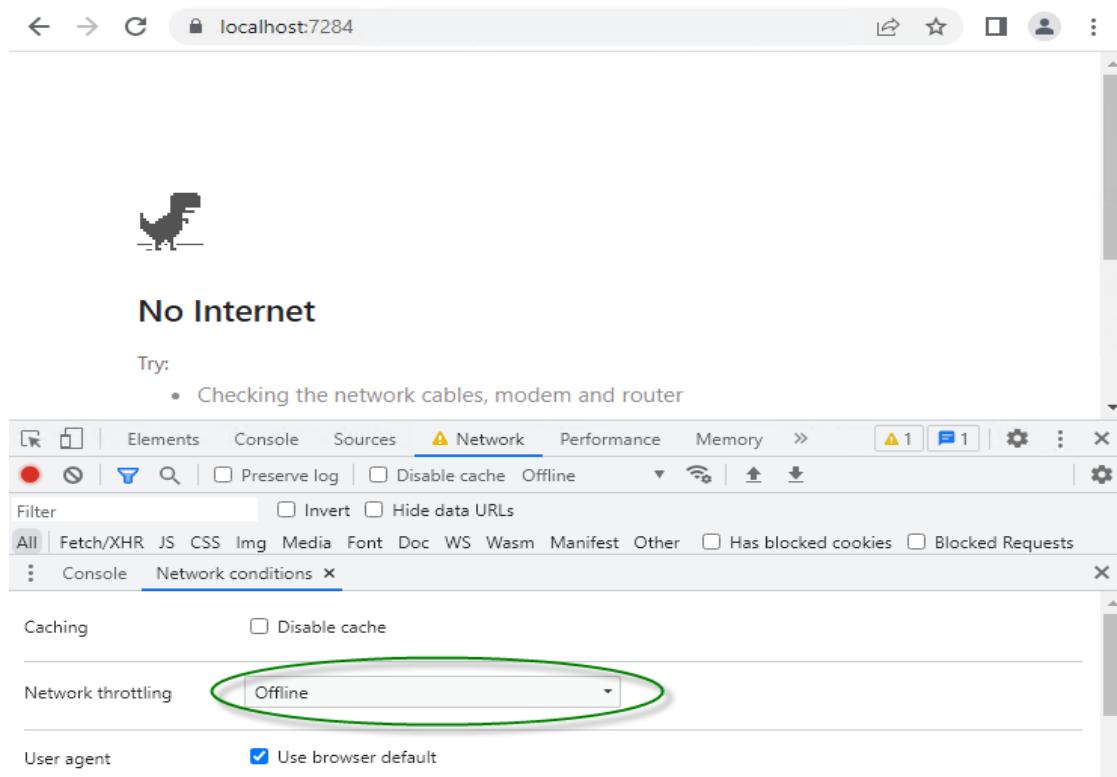
</html>
```

Ponemos a correr la app y vemos que funciona correctamente:



The screenshot shows a Blazor PWA application running locally at localhost:7284. The application's title is "BlazorPWAPrueba". On the left, there's a sidebar with three items: "Home" (selected), "Counter", and "Fetch data". The main content area displays "Hello, world!" and a message: "Welcome to your new app." Below this is a survey prompt: "How is Blazor working for you? Please take our [brief survey](#) and tell us what you think." At the bottom of the main content, there's a network performance timeline from 10 ms to 110 ms. The developer tools Network tab is active, showing a list of requests like "All", "Fetch/XHR", "JS", "CSS", etc., and a detailed timeline below.

Si simulamos estar offline y refrescamos la página, vemos que la app aún no funciona.



The screenshot shows the same Blazor PWA application after simulating an offline environment. The status bar now indicates "No Internet". The main content area displays a "No Internet" message with a small icon of a person walking away. Below this, there's a "Try:" section with a bullet point: "Checking the network cables, modem and router". The developer tools Network tab is still active, but the "Network throttling" dropdown at the bottom is now set to "Offline", which is highlighted with a green oval. Other settings like "Caching" and "User agent" are also visible.

Esto sucede porque por defecto el modo offline está desactivado cuando estamos en un ambiente de desarrollo. Esto lo hacen para que cuando estemos desarrollando no tengamos que estar luchando con la caché del navegador, porque la idea es como veíamos hace un momento que en el ServiceWorker, lo que hacemos es para activar el modo offline, es trabajar con la caché del navegador: guardamos la información de la página en caché y si el usuario no tiene internet, pues utilizamos el caché.

Pero esto es molesto cuando estamos desarrollando, porque pueden ocurrir situaciones en las cuales nos va a salir la información del caché cuando quisieramos no utilizar esa información porque estamos en pleno desarrollo y queremos ver la página actualizada siempre.

Por eso, por defecto, para evitar esa molestia, desactivan el modo offline en desarrollo.

Modo offline en desarrollo

Podemos hacer dos cosas. La primera sería activar el modo offline en desarrollo. La segunda sería publicar nuestro sitio web. Publicaremos el sitio web solamente para que veamos qué sin intervenir de ninguna manera esto va a funcionar.

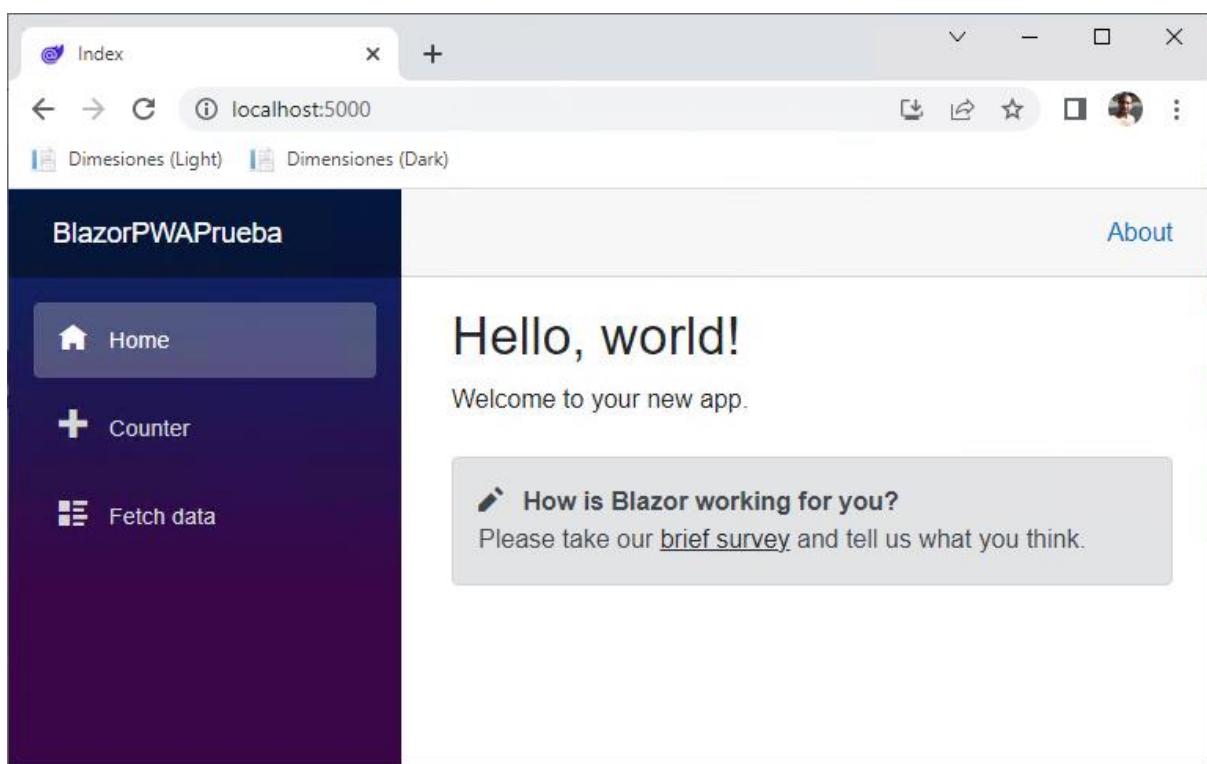
Publish

Click-derecho en el proyecto **Server** y elegimos **Publish**. Elegimos publicar hacia una carpeta. Una vez que nos muestra el perfil creado le damos terminar. Nos muestra el resumen de lo que hará. Presionamos **Publish** para que se publique. Al terminar, nos aparece un botón que dice **Open folder** para dirigirnos a la carpeta donde se publicó.

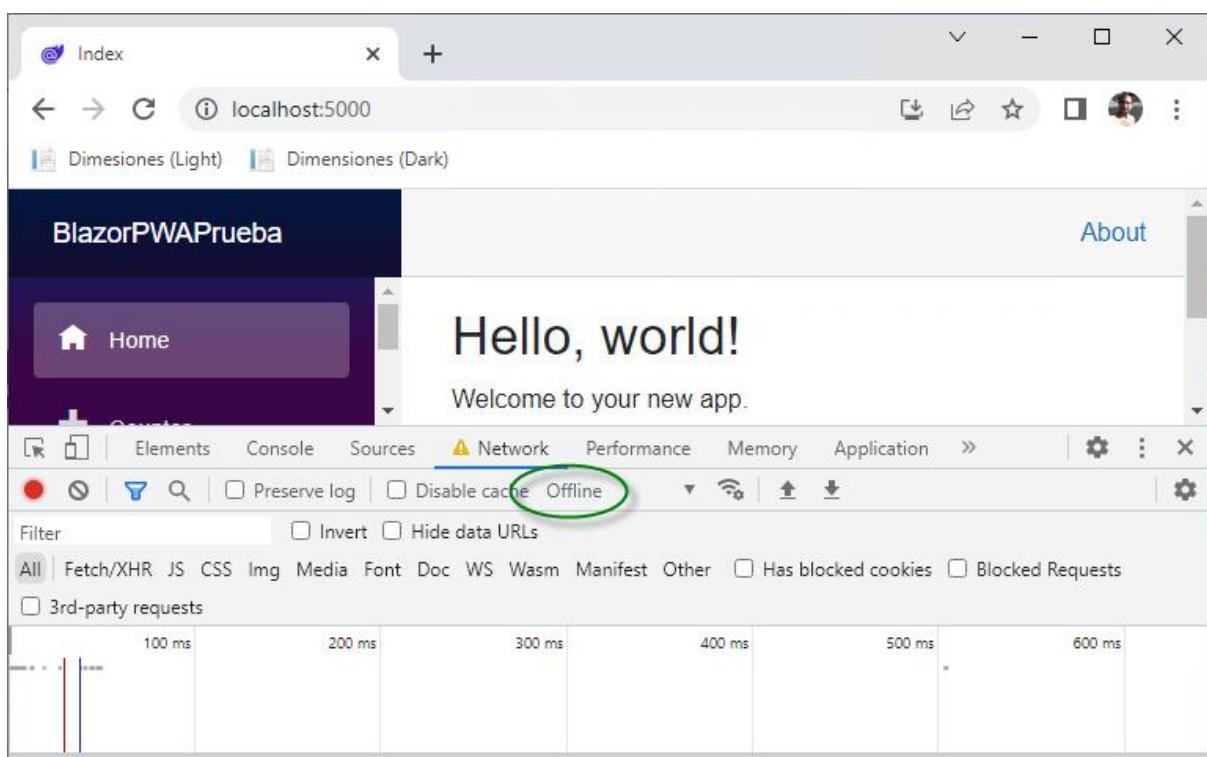
Hacemos **Shift + click-derecho** y elegimos **Open Powershell window here** escribimos donet y el nombre de la DLL y presionamos ENTER.

```
PS C:\Users\EFM\source\repos\BlazorPWAPrueba\Server\bin\Release\net7.0\publish> dotnet BlazorPWAPrueba.Server.dll
[info]: Microsoft.Hosting.Lifetime[14]
Now listening on: http://localhost:5000
[info]: Microsoft.Hosting.Lifetime[0]
Application started. Press Ctrl+C to shut down.
[info]: Microsoft.Hosting.Lifetime[0]
Hosting environment: Production
[info]: Microsoft.Hosting.Lifetime[0]
Content root path: C:\Users\EFM\source\repos\BlazorPWAPrueba\Server\bin\Release\net7.0\publish
```

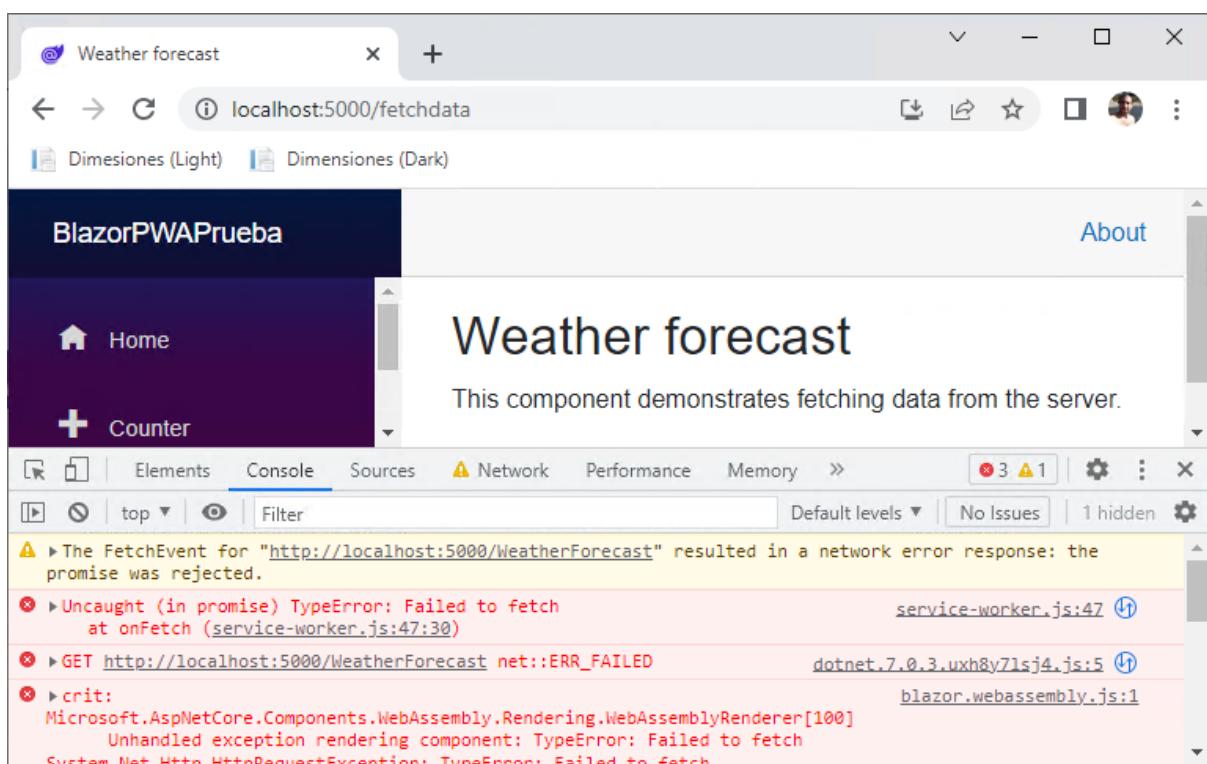
Vemos que la app está corriendo un hosting en <http://localhost:5000>. Abrimos un navegador y nos dirigimos ahí:



Lo ponemos en modo offline y refrescamos:



Casi toda la app funciona, aún en modo offline, menos la parte de Fetch data:



Podemos ver que nos tiró un error de **Failed to fetch**. Esto se debe a que el componente intenta acceder a una página externa para obtener datos de manera online.

```

33     <td>@forecast.Summary</td>
34   </tr>
35 </tbody>
36 </table>
37 }
38
39 @code {
40   private WeatherForecast[]? forecasts;
41
42   protected override async Task OnInitializedAsync()
43   {
44     forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("WeatherForecast");
45   }
46 }
47
48 }
```

La respuesta de esta petición HTTP no está en caché. Lo que está en caché son los archivos estáticos de mi aplicación como archivos DSS, HTML, JavaScript, etcétera, y podemos validar esto yendo al tab de **Application**.

En **Manifest** vemos lo mismo que veíamos antes en el json:

```

{
  "name": "BlazorPWAPrueba",
  "short_name": "BlazorPWAPrueba",
  "description": null,
  "start_url": "http://localhost:5000/",
  "background_color": "#ffffff",
  "display": "standalone"
}

```

En **Service Workers** vemos que hay uno activo y corriendo:

Push	Action
Test push message from DevTools.	<input type="button" value="Push"/>

Sync	Action
test-tag-from-devtools	<input type="button" value="Sync"/>

Periodic Sync	Action
test-tag-from-devtools	<input type="button" value="Periodic Sync"/>

Update Cycle	Version	Activity	Timeline
	#33	Install	
	#33	Wait	
	#33	Activate	██████████

Mientras que en **Cache**, vemos todos los archivos que están cacheados:

#	Name	Response-Type	Status
0	/BlazorPWAPrueba.Client.styles.css	basic	t
1	/_framework/BlazorPWAPrueba.Client.dll	basic	a
2	/_framework/BlazorPWAPrueba.Shared.dll	basic	a
3	/_framework/Microsoft.AspNetCore.Components.Web.dll	basic	a
4	/_framework/Microsoft.AspNetCore.Components.WebAssembly.dll	basic	a
5	/_framework/Microsoft.AspNetCore.Components.dll	basic	a
6	/_framework/Microsoft.Extensions.Configuration.Abstractions.dll	basic	a
7	/_framework/Microsoft.Extensions.Configuration.Json.dll	basic	a
8	/_framework/Microsoft.Extensions.Configuration.dll	basic	a
9	/_framework/Microsoft.Extensions.DependencyInjection.Abstractions.dll	basic	a
10	/_framework/Microsoft.Extensions.DependencyInjection.dll	basic	a

Select a cache entry above t

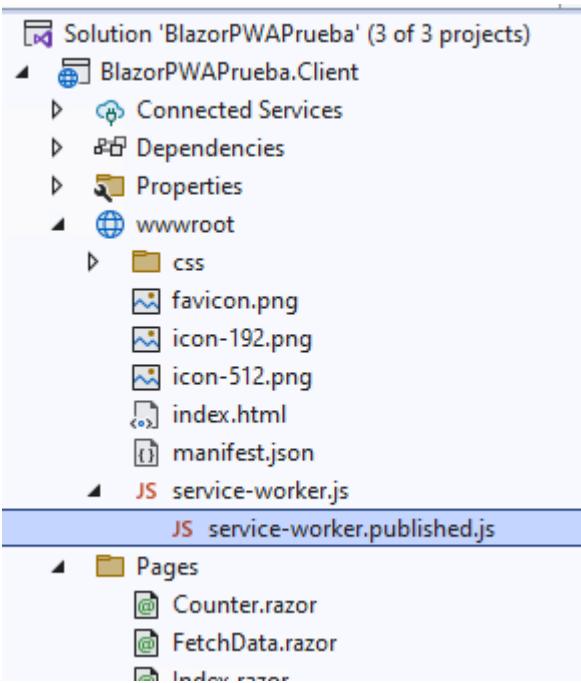
También se cachean otros archivos del framework. Este cache storage que tenemos acá entonces es el que se está utilizando para servir la aplicación al usuario, porque dado el hecho de que supuestamente no tenemos internet, entonces es del caché que viene toda esa información, toda esa data.

Y como no estamos guardando la petición http hacia WeatherForecast, pues por eso el código de ese componente da error.

Cacheando archivos

¿Quién cachea todos los archivos app.css, favicon, DLLs, etc.? Es decir, quien los coloca en el caché. ¿cómo podemos hacer para que nuestros archivos personalizados también nos salgan acá? Esto lo veremos más adelante.

Modo offline



El motivo por el cual el modo offline está deshabilitado es porque es tedioso estar lidiando con el cache del navegador mientras se está en desarrollo.

En el árbol de estructura se puede ver que hay 2 service-workers:

- service-worker.js
- service-worker.published.js

Veamos los códigos de ambos.

service-worker.js

```
// In development, always fetch from the network and do not enable offline support.
// This is because caching would make development more difficult (changes would not
// be reflected on the first load after each change).
self.addEventListener('fetch', () => {});
```

service-worker.published.js

```
// Caution! Be sure you understand the caveats before publishing an application with
// offline support. See https://aka.ms/blazor-offline-considerations

self.importScripts('./service-worker-assets.js');
self.addEventListener('install', event => event.waitUntil(onInstall(event)));
self.addEventListener('activate', event => event.waitUntil(onActivate(event)));
self.addEventListener('fetch', event => event.respondWith(onFetch(event)));

const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const offlineAssetsInclude = [ /\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/, /\.json$/, /\.css$/, /\.woff$/,
/\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/, /\.blat$/, /\.dat$/];
const offlineAssetsExclude = [ /service-worker\.js$/ ];

async function onInstall(event) {
    console.info('Service worker: Install');

    // Fetch and cache all matching items from the assets manifest
```

```

const assetsRequests = self.assetsManifest.assets
    .filter(asset => offlineAssetsInclude.some(pattern => pattern.test(asset.url)))
    .filter(asset => !offlineAssetsExclude.some(pattern => pattern.test(asset.url)))
    .map(asset => new Request(asset.url, { integrity: asset.hash, cache: 'no-cache' }));
    await caches.open(cacheName).then(cache => cache.addAll(assetsRequests));
}

async function onActivate(event) {
    console.info('Service worker: Activate');

    // Delete unused caches
    const cacheKeys = await caches.keys();
    await Promise.all(cacheKeys
        .filter(key => key.startsWith(cacheNamePrefix) && key !== cacheName)
        .map(key => caches.delete(key)));
}

async function onFetch(event) {
    let cachedResponse = null;
    if (event.request.method === 'GET') {
        // For all navigation requests, try to serve index.html from cache
        // If you need some URLs to be server-rendered, edit the following check to exclude those URLs
        const shouldServeIndexHtml = event.request.mode === 'navigate';

        const request = shouldServeIndexHtml ? 'index.html' : event.request;
        const cache = await caches.open(cacheName);
        cachedResponse = await cache.match(request);
    }

    return cachedResponse || fetch(event.request);
}
    
```

Como se ve, el primero está prácticamente vacío. El segundo, en cambio, es el que se utiliza cuando estamos en producción: cuando publicamos la app. Por eso el “published”. Aquí, como podemos ver si hay mucha lógica.

Y en toda esa lógica se encuentra la de utilizar el caché si es posible, o utilizar los recursos del servidor.

Lo primero que vemos es que hay un import a [service-worker-assets.js](#). En este se colocan todos los archivos estáticos de nuestra app. Actualmente luce así:

```

service-worker-assets.js
self.assetsManifest = {
    "assets": [
        {
            "hash": "sha256-IOCs45frNf+D7me6IKLgUhRCdt4VwHCp9H6Ct+YWzfs=",
            "url": "_framework\blazor.webassembly.js"
        },
    ]
}
    
```

```
{  
    "hash": "sha256-DxGvmbtUa6hGJpk3VyMEbkPZ653qWbQkh8X4ZVM9TWM=",  
    "url": "_framework\\dotnet.7.0.3.uxh8y7lsj4.js"  
},  
{  
    "hash": "sha256-jgK4sHqNVMYvwtJ9cbh\\rz\\+ZekC9qEBgHdG\\NsjYhw=",  
    "url": "_framework\\dotnet.timezones.blat"  
},  
{  
    "hash": "sha256-DL4j2DpxtpVPYKs+s2MEt5FbF\\1BGHDewGhsBC8M7as=",  
    "url": "_framework\\dotnet.wasm"  
},  
{  
    "hash": "sha256-tO5O5YzMTVSaKBboxAqezOQL9ewmupzV2JrB5Rkc8a4=",  
    "url": "_framework\\icudt.dat"  
},  
{  
    "hash": "sha256-SZLtQnRc0JkwqHab0UVVP7T3uBPSeYzxzDnpnPpUnHk=",  
    "url": "_framework\\icudt_CJK.dat"  
},  
{  
    "hash": "sha256-8fItetYY8kQ0ww6oxwTLiT3oXIbwHKumbeP2pRF4yTc=",  
    "url": "_framework\\icudt_EFIGS.dat"  
},  
{  
    "hash": "sha256-L7sV7NEYP37\\Qr2FPCePo5cJqRgTXRwGHuwF5Q+0Nfs=",  
    "url": "_framework\\icudt_no_CJK.dat"  
},  
{  
    "hash": "sha256-lyAu9jYllrolkX2w+TXrsevJ9Jibo9yyv4jGpZuHQRY=",  
    "url": "_framework\\blazor.boot.json"  
},  
{  
    "hash": "sha256-UyaPY35IA+UWlzfX6mOwVE1oX7LiphPnB\\Me5R0dTGg=",  
    "url": "_framework\\BlazorPWAPrueba.Client.dll"  
},  
{  
    "hash": "sha256-11ApxHMuE9XU5ZxT9SBpeCuq4nCGygMkEKLwRzf2tLs=",  
    "url": "_framework\\BlazorPWAPrueba.Shared.dll"  
},  
{  
    "hash": "sha256-lhwBz44Bz4YHLQFXbXFz0VYvCp2ahLGOBoi+e6ZhOQE=",  
    "url": "_framework\\Microsoft.AspNetCore.Components.dll"  
},  
{  
    "hash": "sha256-0yCcULHtWBrHOqJji1m34+Xd+S67k7NMo\\JdsZfvcpM=",  
    "url": "_framework\\Microsoft.AspNetCore.Components.Web.dll"  
},  
{  
    "hash": "sha256-VoOExk2fEE249K3Jayw1Nymq9Bc8LIKEWiOG9\\0GiNY=",  
}
```

```
"url": "_framework\Microsoft.AspNetCore.Components.WebAssembly.dll"
},
{
"hash": "sha256-X\f4fDl2culRXeWHhKV\f2UqQbFioD+RU4a4CEh0zrrQ=",
"url": "_framework\Microsoft.Extensions.Configuration.Abstractions.dll"
},
{
"hash": "sha256-DBOKSPriP2JDxVbbWrLXyD3K4\vx3RBifNBWk\q1I39M=",
"url": "_framework\Microsoft.Extensions.Configuration.dll"
},
{
"hash": "sha256-nTqLKuydttqLtE3VT3p6XhPmLxuaJDd0cL3Qzt8D4Ro=",
"url": "_framework\Microsoft.Extensions.Configuration.Json.dll"
},
{
"hash": "sha256-tkdkrAT4n35\Yaa5YMYJpqdzzpHol9Ox3jP0FAYcFPE=",
"url": "_framework\Microsoft.Extensions.DependencyInjection.Abstractions.dll"
},
{
"hash": "sha256-qi0kE7rp0kdsNqdL6DyPZEeimjUGvcLT4iWQX0YnRus=",
"url": "_framework\Microsoft.Extensions.DependencyInjection.dll"
},
{
"hash": "sha256-x+N2sgSEct\XXqose+TJ\8DQXHbcSLTTD03Y22EXT8=",
"url": "_framework\Microsoft.Extensions.Logging.Abstractions.dll"
},
{
"hash": "sha256-oEPPw1EPpaOHv+EHwoT2WcgDiRbjEXxigiW7V44clYE=",
"url": "_framework\Microsoft.Extensions.Logging.dll"
},
{
"hash": "sha256-WPBQk7rt2p4aMEo2pTp1sBpsiWdE8MmWuIYq+zI1ceo=",
"url": "_framework\Microsoft.Extensions.Options.dll"
},
{
"hash": "sha256-eXvGx2jcjpTPEJoAHBsW\VuMPbNyyU+AsuhPmkzSSRY=",
"url": "_framework\Microsoft.Extensions.Primitives.dll"
},
{
"hash": "sha256-L85W0kGe1vzXNUYnqQARFyJyU5KutJYw6dMeZkGsS6Q=",
"url": "_framework\Microsoft.JSInterop.dll"
},
{
"hash": "sha256-hUxOrW0i2d4NV\3At6wRqsDMpya9fFCtdE6YcQJb8d40=",
"url": "_framework\Microsoft.JSInterop.WebAssembly.dll"
},
{
"hash": "sha256-p\W+WSIkL9jzyEasd03baj2LYf3Kyz9kwWgFTHUm\VSs=",
"url": "_framework\System.Collections.Concurrent.dll"
},
```

```
{  
    "hash": "sha256-Dr\\xQMK+1VDuOEur4pePjCX7RGN+pjOOqE8IMEndiLY=",  
    "url": "_framework\\System.Collections.dll"  
},  
{  
    "hash": "sha256-i8l1LCCL8JFSIBljON1cA2eYNziHWE21TXOhsLVQa+Y=",  
    "url": "_framework\\System.ComponentModel.dll"  
},  
{  
    "hash": "sha256-waeNUqVloRgeBajB+rbwHe6iJwRTvmoPTzYhltCisek=",  
    "url": "_framework\\System.Linq.dll"  
},  
{  
    "hash": "sha256-gDaKbsx6vP0iZXZw2UKct1OanzfmCr3BInmcHEEbdtQ=",  
    "url": "_framework\\System.Memory.dll"  
},  
{  
    "hash": "sha256-1jSpLAgNSzEkAbiBh6lrlfLU1BaoCyO4g8gsiJXVd0=",  
    "url": "_framework\\System.Net.Http.dll"  
},  
{  
    "hash": "sha256-XnkC\\gRnSOcysCqjfScengM\\WP9\\Kj7m5YnHcshq2VE=",  
    "url": "_framework\\System.Net.Http.Json.dll"  
},  
{  
    "hash": "sha256-ajFzrovIcALXAxAyU2LKbT31dShjCOcmkdrHJDn3lbfo=",  
    "url": "_framework\\System.Net.Primitives.dll"  
},  
{  
    "hash": "sha256-Cn8gdeGWELeUc2OK3Pi+coEv+11rLXvmzAYm7bh0YVc=",  
    "url": "_framework\\System.Private.CoreLib.dll"  
},  
{  
    "hash": "sha256-optr\\BWWWhT5ZahDA3AZG6HOMSpOms3u\\rMEM+WdVjDU=",  
    "url": "_framework\\System.Private.Uri.dll"  
},  
{  
    "hash": "sha256-6CIDCzQ42FIMxfZ4hiOjqtLjudjomWavCgNvPCBT\\os=",  
    "url": "_framework\\System.Runtime.dll"  
},  
{  
    "hash": "sha256-lhmhXzzO60OoH+wLFM5KuYihvi5jEDXvliOZzVV/5i4o=",  
    "url": "_framework\\System.Runtime.InteropServices.JavaScript.dll"  
},  
{  
    "hash": "sha256-hnmkcWS1u4akDjNnEVBY3N9HGT+VmFajs8yg5AdB2mo=",  
    "url": "_framework\\System.Text.Encoding.Web.dll"  
},  
{  
    "hash": "sha256-e2PIYnbE9oHb01tl7IFI0ATSF9y3OK9oUNzLj4IVH4M=",  
}
```

```
"url": "_framework\System.Text.Json.dll"
},
{
"hash": "sha256-5IpymjlTtTF3Qny8ZOOG6BpFEuUKUJvDlj7UwGbm2T4=",
"url": "BlazorPWAPrueba.Client.styles.css"
},
{
"hash": "sha256-uhotZszkBLq\V8xt8UtpU6IGHElqbqLsFUVGyeIV2TU=",
"url": "css\app.css"
},
{
"hash": "sha256-z8OR40MowJ8GgK6P89Y+hiJK5+cclzFHzLhFQLL92bg=",
"url": "css\bootstrap\bootstrap.min.css"
},
{
"hash": "sha256-gBwg2tmA0Ci2u54gMF1jNCVku6vznarkLS6D76htNNQ=",
"url": "css\bootstrap\bootstrap.min.css.map"
},
{
"hash": "sha256-BJV/G+e+y7bQdrYkS2RBTyNfBHpA9IuGaPmf9htub5MQ=",
"url": "css\open-iconic\font\css\open-iconic-bootstrap.min.css"
},
{
"hash": "sha256-OK3poGPgzKI2NzNgP07XMbJa3Dz6USoUh\chSkSvQpc=",
"url": "css\open-iconic\font\fonts\open-iconic.eot"
},
{
"hash": "sha256-sDUtuZAEzWZyv\U1xl\9D3ehyU69JE+FvAcu5HQ+Va0=",
"url": "css\open-iconic\font\fonts\open-iconic.otf"
},
{
"hash": "sha256-+P1oQ5jPzOVJGC52E1oxGXIXxxCyMIqy6A9cNxGYzVk=",
"url": "css\open-iconic\font\fonts\open-iconic.svg"
},
{
"hash": "sha256-p+RP8CV3vRK1YblkNzq3rPo1jyETPnR07ULb+HVYL8w=",
"url": "css\open-iconic\font\fonts\open-iconic.ttf"
},
{
"hash": "sha256-cZPqVIRJfSNW0KaQ4+UPOXZ\vvQzXlejRDwUNdZlofl=",
"url": "css\open-iconic\font\fonts\open-iconic.woff"
},
{
"hash": "sha256-jA4J4h\k76zVxbFKEaWwFKJccmO0voOQ1DbUW+5YNII=",
"url": "css\open-iconic\FONT-LICENSE"
},
{
"hash": "sha256-aF5g\izareSj02F3MPSoTGNbcMBI9nmZKDe04zjU\ss=",
"url": "css\open-iconic\ICON-LICENSE"
},
```

```
{
    "hash": "sha256-waukoLqsilAw\nnXpsKkTHnhImmcPijcBEd2lqzJNN0=",
    "url": "css\open-iconic\README.md"
},
{
    "hash": "sha256-4mWsDy3aHI36ZbGt8zByK7Pvd4kRUoNgTYzRnwmPHwg=",
    "url": "favicon.png"
},
{
    "hash": "sha256-DbpQaq68ZSb5IoPosBErM1QWBfsbTxpJqhU0REi6wP4=",
    "url": "icon-192.png"
},
{
    "hash": "sha256-oEo6d+KqX5fjxTiZk/w9NB3Mi0+ycS5yLwCKwr4IkA=",
    "url": "icon-512.png"
},
{
    "hash": "sha256-vA+f1JeWz9MEz3iLdUWyM9sluTpQiYJa0X0lnI9Xwk=",
    "url": "index.html"
},
{
    "hash": "sha256-VS7CzVPTFteNsFYNL/RwlwWIjR1ORGcEvIgrzPALSTc=",
    "url": "manifest.json"
}
],
"version": "fro3SnUP"
};
```

Crearemos el archivo **Utilities.js** en la carpeta **wwwroot** (sin código ya que no es necesario) y re-publicamos. Supuestamente luego de esto, debería aparecer en service-worker-assets.js estaría el nuevo archivo, pero no apareció.

Con esto lograríamos que nuestro **service-worker.published.js** ejecutará el evento **onInstall** (cuando se está instalando). Cuando se ingresa a una página que tiene SW se dan 2 eventos: **onInstall** que es cuando se está instalando el SW y **onActivate** que es cuando se activa el SW instalado.

Cuando un SW está activado, el usuario debe salir del navegador y volver a entrar para que se detecte una nueva versión del SW (si existiera) y la instalara. A menos, que el usuario force la instalación del nuevo SW con las herramientas del Chrome.

En el código de **service-worker.published.js**, tenemos assetsRequests y de aquí estamos diciendo **assetsManifest** y aplicando un filtro donde estamos tomando de ese conjunto de archivos que tenemos acá (en service-worker-assets.js declaramos el objeto **self.assetsManifests**).

onInstall

service-worker.published.js

```
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const offlineAssetsInclude = [ /\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/, /\.json$/, /\.css$/, /\.woff$/,
/\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/];
const offlineAssetsExclude = [/^service-worker\.js$/];

// Fetch and cache all matching items from the assets manifest
const assetsRequests = self.assetsManifest.assets
    .filter(asset => offlineAssetsInclude.some(pattern => pattern.test(asset.url)))
    .filter(asset => !offlineAssetsExclude.some(pattern => pattern.test(asset.url)))
    .map(asset => new Request(asset.url, { integrity: asset.hash, cache: 'no-cache' }));
await caches.open(cacheName).then(cache => cache.addAll(assetsRequests));
}
```

A ese objeto, le aplicamos el filtro de tal manera que solamente guardemos en caché aquellos que queramos que se guarden en caché. Por ejemplo, aquí nos dicen que se van a incluir todos los archivos que sean de DLL, PDB, WASM, HTML, JS, CSS, etc.

Eso explica por qué Utilities.js sería servido en el caché. Luego se están mapeando hacia un nuevos request y luego se está abriendo el caché llamado **cacheName** y se está incluyendo todos estos archivos, la información, el contenido de los archivos en el caché. Vemos que **cacheName** es un nombre autogenerado, que se va a ir cambiando a medida que vayamos desarrollando nuestra aplicación.

De esta manera no vamos a tener un caché anticuado con versiones antiguas de nuestros archivos.

onActivate

service-worker.published.js

```
const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const offlineAssetsInclude = [ /\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/, /\.json$/, /\.css$/, /\.woff$/, 
  /\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/ , /\ blat$/ , /\ dat$/ ];
const offlineAssetsExclude = [ /^service-worker\.js$/ ];

async function onActivate(event) {
  console.info('Service worker: Activate');

  // Delete unused caches
  const cacheKeys = await caches.keys();
  await Promise.all(cacheKeys
    .filter(key => key.startsWith(cacheNamePrefix) && key !== cacheName)
    .map(key => caches.delete(key)));
}
```

En **onActivate** lo que se hace es que se borran los cachés que ya no están en uso, es decir, versiones anteriores del caché.

onFetch

¿cómo funciona eso de que se está utilizando el caché en vez de realizar una petición HTTP hacia el servidor?

Pues para eso tenemos el **onFetch**.

service-worker.published.js

```
self.addEventListener('fetch', event => event.respondWith(onFetch(event)));

const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const offlineAssetsInclude = [ /\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/, /\.json$/, /\.css$/, /\.woff$/,
  /\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/ , /\ blat$/ , /\ dat$/ ];
const offlineAssetsExclude = [ /^service-worker\.js$/ ];

async function onFetch(event) {
  let cachedResponse = null;
  if (event.request.method === 'GET') {
    // For all navigation requests, try to serve index.html from cache
    // If you need some URLs to be server-rendered, edit the following check to exclude those URLs
    const shouldServeIndexHtml = event.request.mode === 'navigate';

    const request = shouldServeIndexHtml ? 'index.html' : event.request;
    const cache = await caches.open(cacheName);
    cachedResponse = await cache.match(request);
  }
}
```

Si venimos acá arriba tenemos un **addEventListener** de **Fetch**, lo que quiere decir que cada vez que se haga un fetch, es decir una petición HTTP desde nuestra aplicación hacia un servidor se va a ejecutar este método onFetch. Éste, es lo que hace lo siguiente: primero define un **cacheResponse** y lo setea a **null**.

Si el método es **get**, entonces entra en el cuerpo del IF en el que básicamente revisa si la petición fetch es de modo de navegación (como cuando seguimos un link). Es diferente cuando tenemos un código de JavaScript, que es una petición hacia el HTTP, hacia un web API para obtener un resultado JSON por ejemplo.

Luego vemos si lo que debemos servir es el index.html o si debemos continuar con el request y luego abrimos el cache, verificamos el request (de la línea anterior y vemos si el caché (**cacheResponse**) no es nulo, pues enviamos esto (**cacheResponse**)

Es decir, que estamos utilizando una estrategia en la cual estamos dando prioridad a lo que tenemos en caché sobre lo que está en el servidor web.

Ahora, si **cacheResponse** es nulo, lo que quiere decir que lo que estamos buscando no está en caché, entonces utilizamos **fetch(evento.request)**, lo que quiere decir sigue hacia adelante con la petición HTTP que estábamos haciendo.

Eso es lo que sucede, por ejemplo, con nuestro FetchData. En ese componente estamos haciendo una petición hace HTTP. Lo primero que hace el SW es verificar si el resultado de esa petición está en caché. Si estuviesen caché se nos retorna eso.

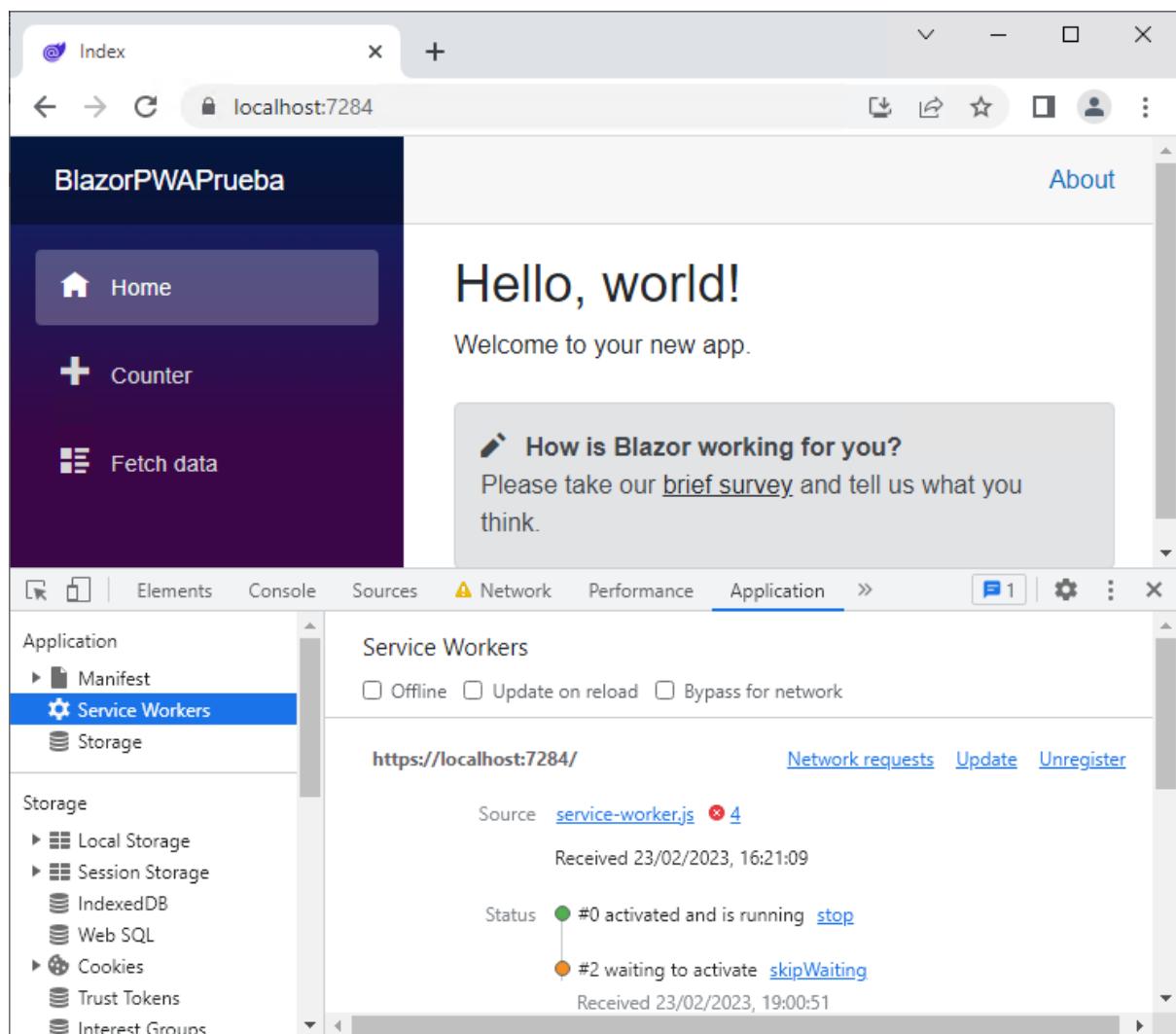
Como no está en caché, entonces se procede a ir hacia el servidor web para poder satisfacer esa petición HTTP Y eso se hace con **fetch(evento.request)**.

Así que esta es la manera en la cual funciona la estrategia de caché por defecto que tenemos en **Blazor**.

Si queremos hacer pruebas del SW en tiempo de desarrollo, es tan sencillo como tomar todo el código de **service-worker-published.js**, copiarlo y pegarlo en **service-worker.js**.

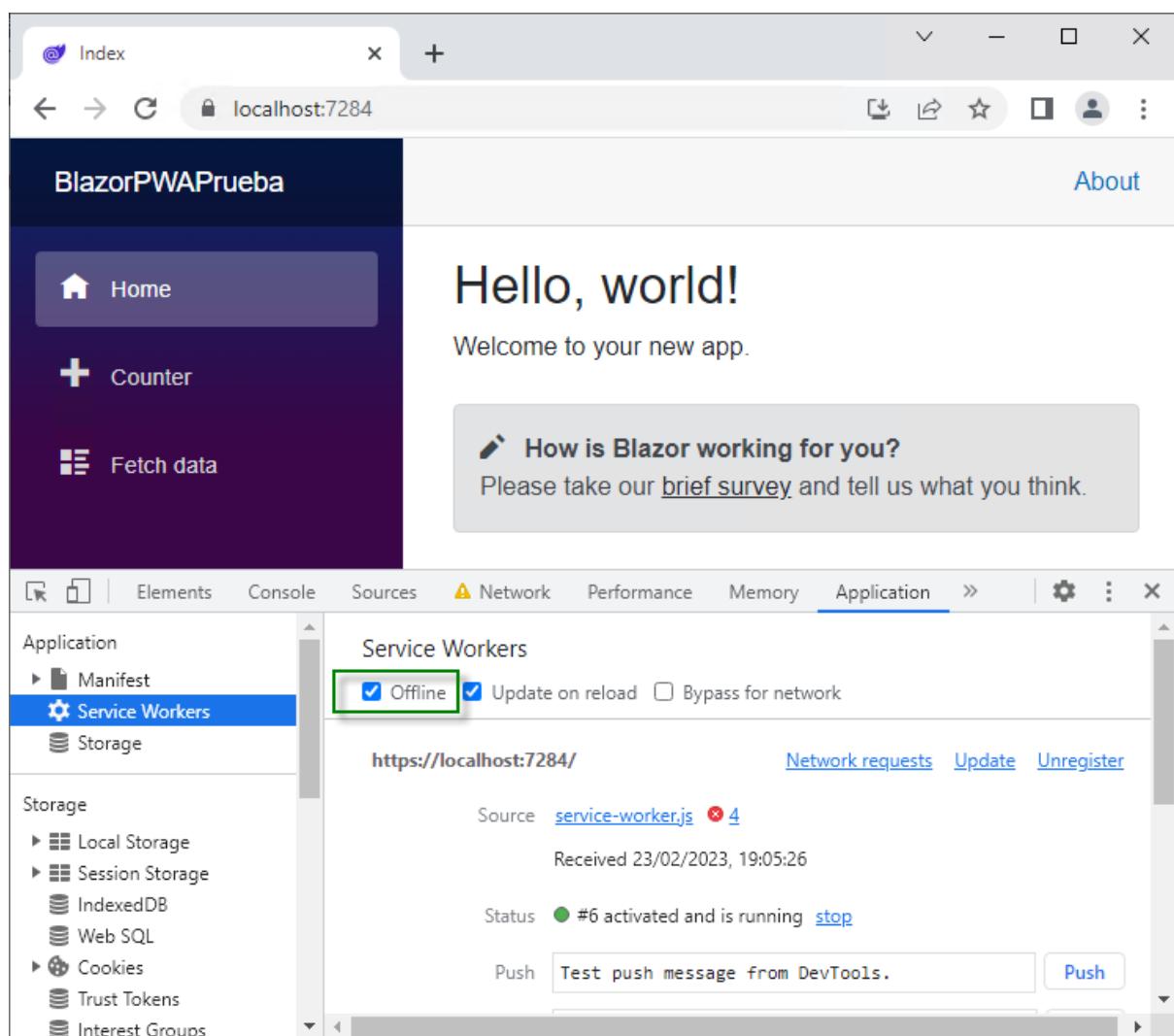
Ya con eso tenemos la función olvidad de cache en modo de desarrollo.

Hacemos este cambio y corremos de nuevo:



Vemos que hay un nuevo SW esperando para instalarse. Vemos que ya está instalado pero no se ha activado. Podemos utilizar la herramienta de **skipWaiting** para que se active la nueva versión. También se puede usar el checkbox de **Update on reload** para que cada vez que refresque active un SW si existiera.

Lo ponemos Offline y refreshcamos:



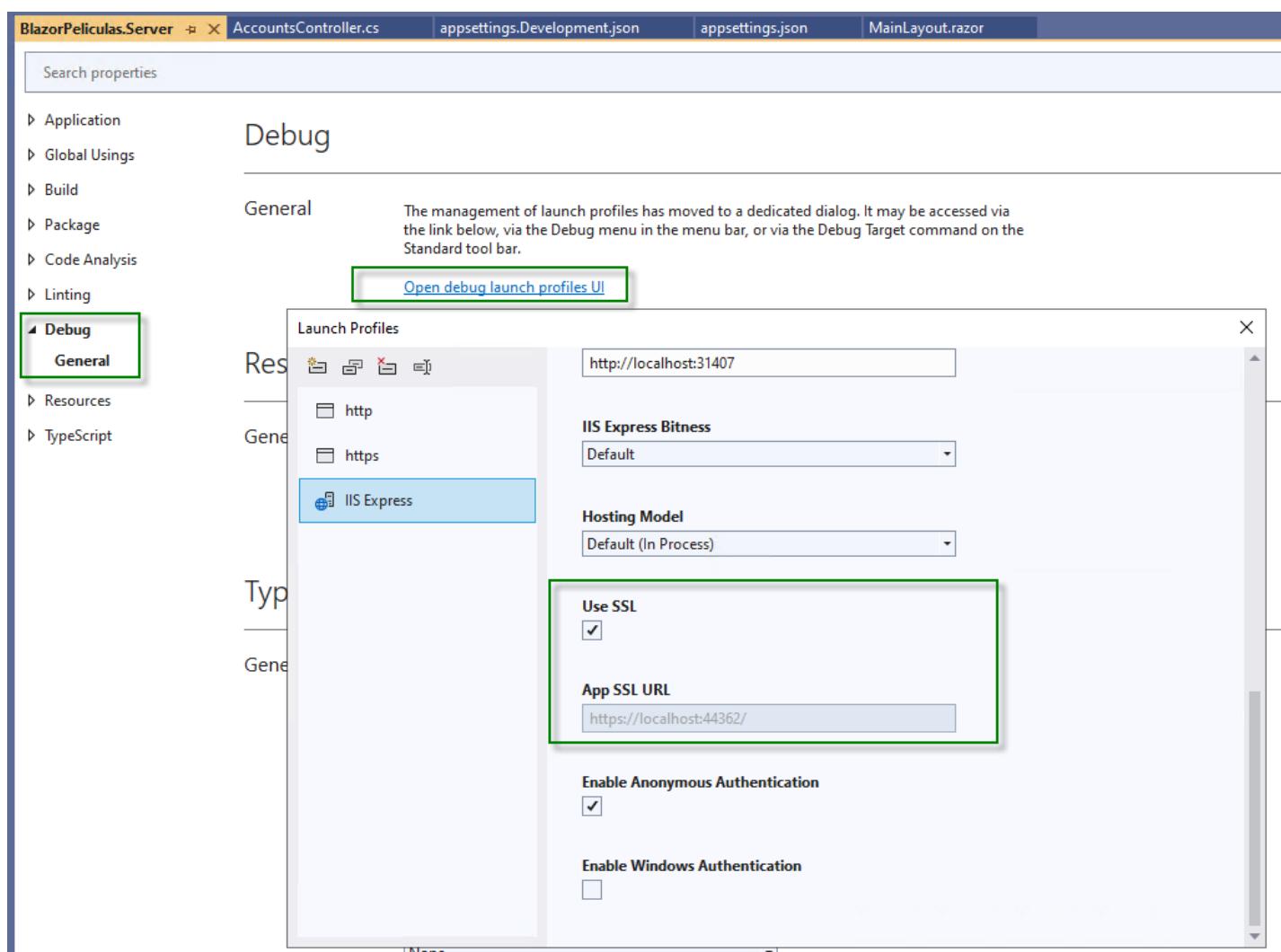
The screenshot shows a Blazor PWA application running in a browser. The sidebar on the left contains links for 'Home', 'Counter', and 'Fetch data'. The main content area displays 'Hello, world!' and 'Welcome to your new app.' Below this is a survey message: 'How is Blazor working for you? Please take our [brief survey](#) and tell us what you think.' The browser's developer tools are open, specifically the Application tab. In the Service Workers section, the 'Offline' checkbox is selected and highlighted with a green border. The source file listed is 'service-worker.js' with 4 errors. The status shows that worker #6 is activated and running. There is a 'Push' button with a text input field containing 'Test push message from DevTools.' and a 'Push' button.

Vemos que la app sigue operativa.

Transformar app en PWA

Sólo para quedarnos con una copia de ambos, copiaremos la carpeta **BlazorPelículas** como **BlazorPelículasPWA** y abrimos la solución de esta última carpeta.

El primer paso es asegurarnos que esté configurado para usar HTTPS ya que es un requisito indispensable para PWAs. Click-derecho en el proyecto **Server** y elegimos **Properties**. Vamos a la pestaña **Debug** y seguimos el link para acceder a la info dedicada de la sección. Vemos que está habilitado el uso de **SSL**



Copiamos la estructura del proyecto **BlazorPWAPrueba**. Vamos a la carpeta **Client\wwwroot** y copiamos los siguientes archivos:

	css	23/02/2023 16:00	File folder	
	favicon.png	23/02/2023 16:00	PNG File	2 KB
	icon-192.png	23/02/2023 16:00	PNG File	3 KB
	icon-512.png	23/02/2023 16:00	PNG File	7 KB
	index.html	23/02/2023 16:00	Chrome HTML Do...	2 KB
	manifest.json	23/02/2023 16:00	JSON Source File	1 KB
	service-worker.js	23/02/2023 19:00	JavaScript File	3 KB
	service-worker.published.js	23/02/2023 16:00	JavaScript File	3 KB
	Utilities.js	23/02/2023 18:23	JavaScript File	1 KB

Pegamos los archivos en **BlazorPeliculasPWA\Client\wwwroot**.

Abrimos el index.html y agregamos el manifest, el serviceworker y los íconos:

index.html

```

<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title>BlazorPeliculas</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
    <link rel="icon" type="image/png" href="favicon.png" />
    <link href="BlazorPeliculas.Client.styles.css" rel="stylesheet" />
    <link href="_content/Blazored.Typeahead/blazored-typeahead.css" rel="stylesheet" />
    <link href="manifest.json" rel="manifest" />
    <link rel="apple-touch-icon" sizes="512x512" href="icon-512.png" />
    <link rel="apple-touch-icon" sizes="192x192" href="icon-192.png" />
</head>

<body>
    <div id="app">
        <svg class="loading-progress">
            <circle r="40%" cx="50%" cy="50%" />
            <circle r="40%" cx="50%" cy="50%" />
        </svg>
        <div class="loading-progress-text"></div>
    </div>

    <div id="blazor-error-ui">
        An unhandled error has occurred.
    </div>

```

```
<a href="" class="reload">Reload</a>
<a class="dismiss">X </a>
</div>
<script src="_framework/blazor.webassembly.js"></script>
<script>navigator.serviceWorker.register('service-worker.js');</script>
<script src="js/Utilities.js"></script>
<script src="_content/CurrieTechnologies.Razor.Sweetalert2/sweetalert2.min.js"></script>
<script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>
</body>

</html>
```

Otra cosa que hay que hacer es la lógica de **service-worker-assets.js** y la lógica de que cuando publiquemos se use el **service-worker-published.js** y no el **service-worker.js**.

Para eso, hacemos click-derecho sobre **Client** y elegimos **Edit Project File** tanto en el proyecto **BlazorPeliculas** como en el de **BlazorPeliculasPWA**.

La primera línea de código resaltada es la que se encarga de compilar nuestro proyecto, verificar los archivos estáticos y ver que los tenemos que colocar en ese archivo **service-worker-assets.js**.

También agregamos el ItemGroup para que haga el cambio de SW al momento de publicar.

BlazorPeliculas.Client.csproj

```
<Project Sdk="Microsoft.NET.Sdk.BlazorWebAssembly">

<PropertyGroup>
  <TargetFramework>net7.0</TargetFramework>
  <Nullable>enable</Nullable>
  <ImplicitUsings>enable</ImplicitUsings>
  <ServiceWorkerAssetsManifest>service-worker-assets.js</ServiceWorkerAssetsManifest>
</PropertyGroup>

<ItemGroup>
  <PackageReference Include="Blazored.Typeahead" Version="4.7.0" />
  <PackageReference Include="CurrieTechnologies.Razor.Sweetalert2" Version="5.4.0" />
  <PackageReference Include="Markdig" Version="0.30.4" />
  <PackageReference Include="MathNet.Numerics" Version="5.0.0" />
  <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly" Version="7.0.3" />
  <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.Authentication" Version="7.0.3" />
  <PackageReference Include="Microsoft.AspNetCore.Components.WebAssembly.DevServer" Version="7.0.3" PrivateAssets="all" />
  <PackageReference Include="System.IdentityModel.Tokens.Jwt" Version="6.26.1" />
</ItemGroup>
```

```
<ItemGroup>
  <ProjectReference Include=".\\Shared\\BlazorPeliculas.Shared.csproj" />
</ItemGroup>

<ItemGroup>
  <BlazorWebAssemblyLazyLoad Include="MathNet.Numerics.dll" />
</ItemGroup>

<ItemGroup>
  <ServiceWorker Include="wwwroot\\service-worker.js" PublishedContent="wwwroot\\service-
worker.published.js" />
</ItemGroup>
</Project>
```

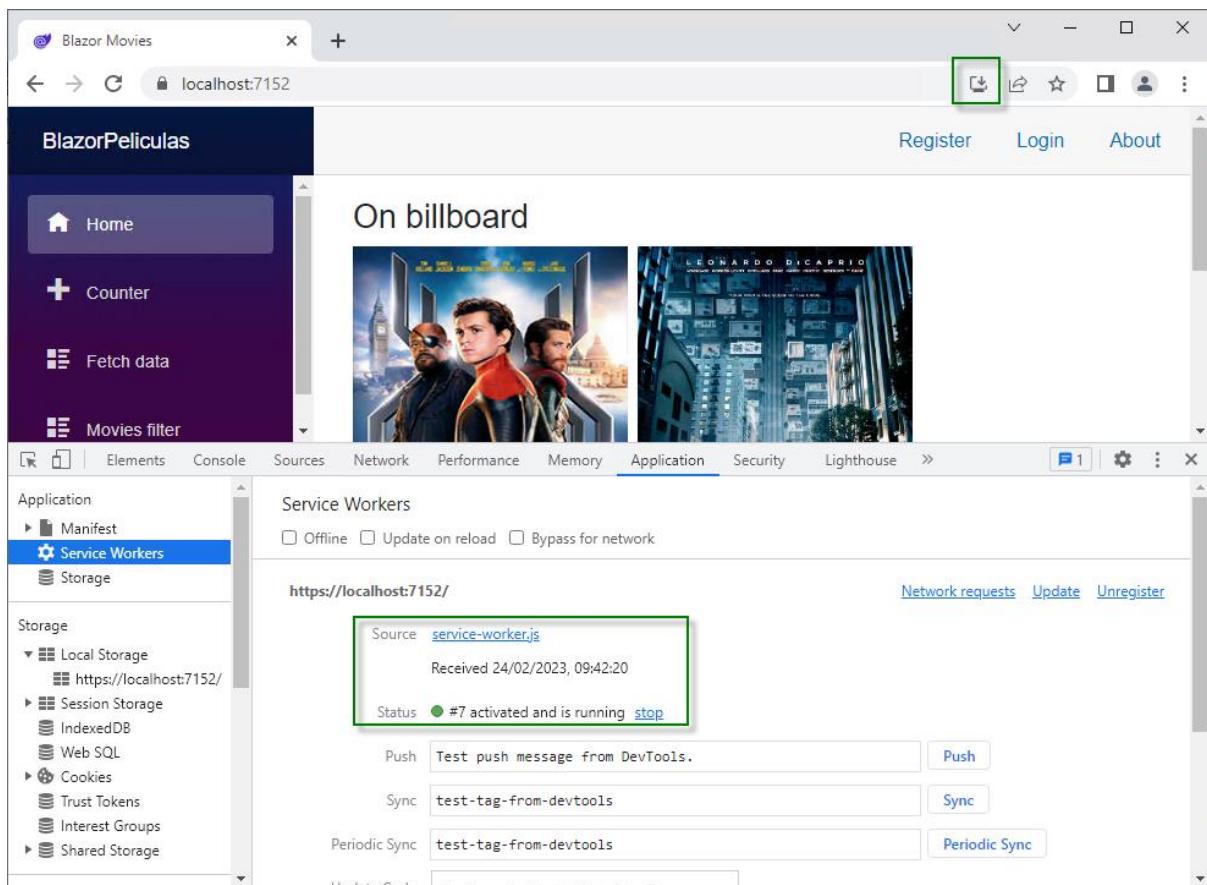
También debemos cambiar el nombre en el manifiesto.

Manifest.json

```
{
  "name": "BlazorPeliculas",
  "short_name": "BlazorPeliculas",
  "start_url": "./",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#03173d",
  "prefer_related_applications": false,
  "icons": [
    {
      "src": "icon-512.png",
      "type": "image/png",
      "sizes": "512x512"
    },
    {
      "src": "icon-192.png",
      "type": "image/png",
      "sizes": "192x192"
    }
  ]
}
```

Obviamente, si quisieramos podríamos cambiar el ícono que se usaría en la aplicación.

Hacemos un **Clean Solution** y **Rebuild Solution** (por las dudas) y corremos la app. Vemos que la app instala su SW y lo activa. También se ve que Chrome muestra el ícono para instalar la APP (al escritorio) próximo a la barra de direcciones:

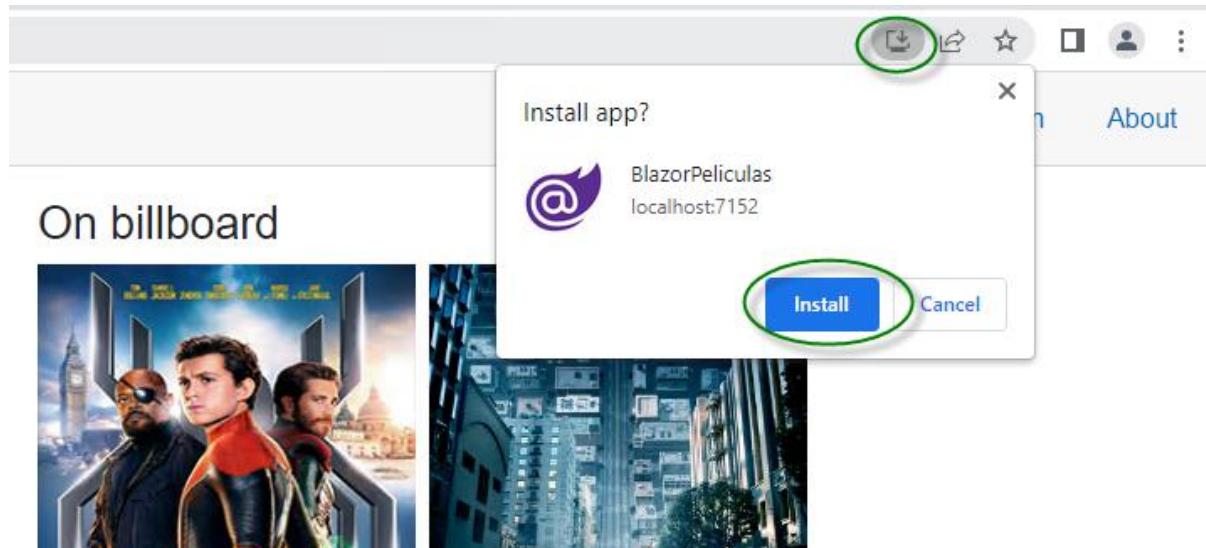


Vemos que también tenemos el cache con todos los archivos necesario para correr de manera offline.

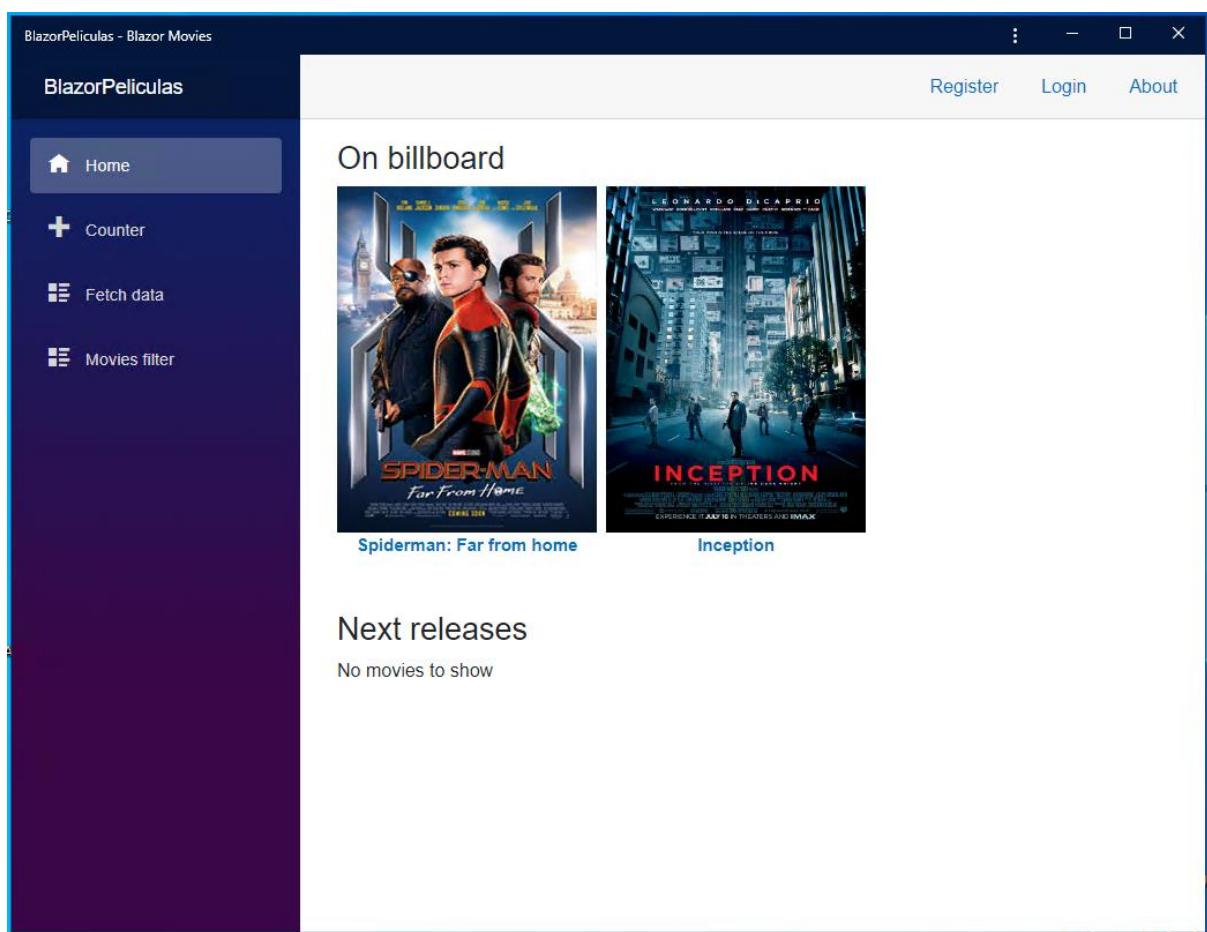
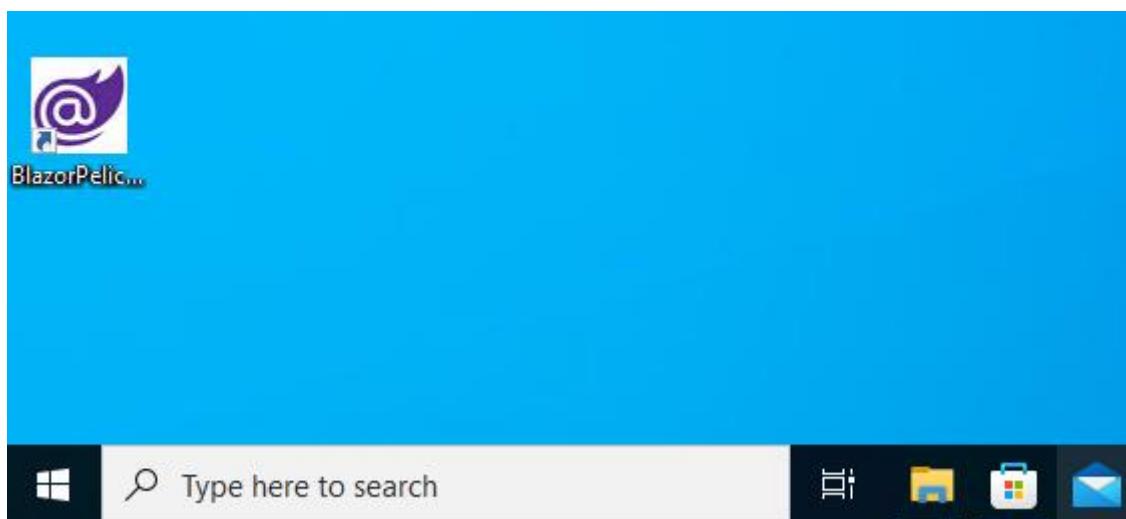
The screenshot shows a browser window for "Blazor Movies" at localhost:7152. On the left, a sidebar menu includes "Home", "Counter", "Fetch data", and "Movies filter". The main content area displays a movie poster for "Spider-Man: Far From Home" with the title "On billboard". At the bottom, the browser's developer tools Network tab is open, showing a table of cached resources. The table has columns for #, Name, Response-T..., Content-Type, Content-Le..., Time Cached, and Vary Header. The first few rows show files like "/BlazorPeliculas.Client.styles.css", "/_content/Blazored.Typeahead/blazored-typeahead.css", and "/_content/Blazored.Typeahead/blazored-typeahead.js". A green circle highlights the "Install app?" button in the top right corner of the browser window.

#	Name	Response-T...	Content-Type	Content-Le...	Time Cached	Vary Header
0	/BlazorPeliculas.Client.styles.css	basic	text/css		3,327	24/02/2023,...
1	/_content/Blazored.Typeahead/blazored-typeahead.css	basic	text/css		5,339	24/02/2023,...
2	/_content/Blazored.Typeahead/blazored-typeahead.js	basic	text/javascript		2,932	24/02/2023,...
3	/_content/CurrieTechnologies.Razor.SweetAlert2/bootstrap...	basic	text/css		29,338	24/02/2023,...
4	/_content/CurrieTechnologies.Razor.SweetAlert2/bootstrap...	basic	text/javascript		148	24/02/2023,...
5	/_content/CurrieTechnologies.Razor.SweetAlert2/bootstrap...	basic	text/css		29,338	24/02/2023,...
6	/_content/CurrieTechnologies.Razor.SweetAlert2/bootstrap...	basic	text/javascript		0	24/02/2023,...
7	/_content/CurrieTechnologies.Razor.SweetAlert2/borderles...	basic	text/css		20,733	24/02/2023,...
8	/_content/CurrieTechnologies.Razor.SweetAlert2/borderles...	basic	text/javascript		148	24/02/2023,...
9	/_content/CurrieTechnologies.Razor.SweetAlert2/borderles...	basic	text/css		20,733	24/02/2023,...
10	/_content/CurrieTechnologies.Razor.SweetAlert2/borderles...	basic	text/javascript		0	24/02/2023,...

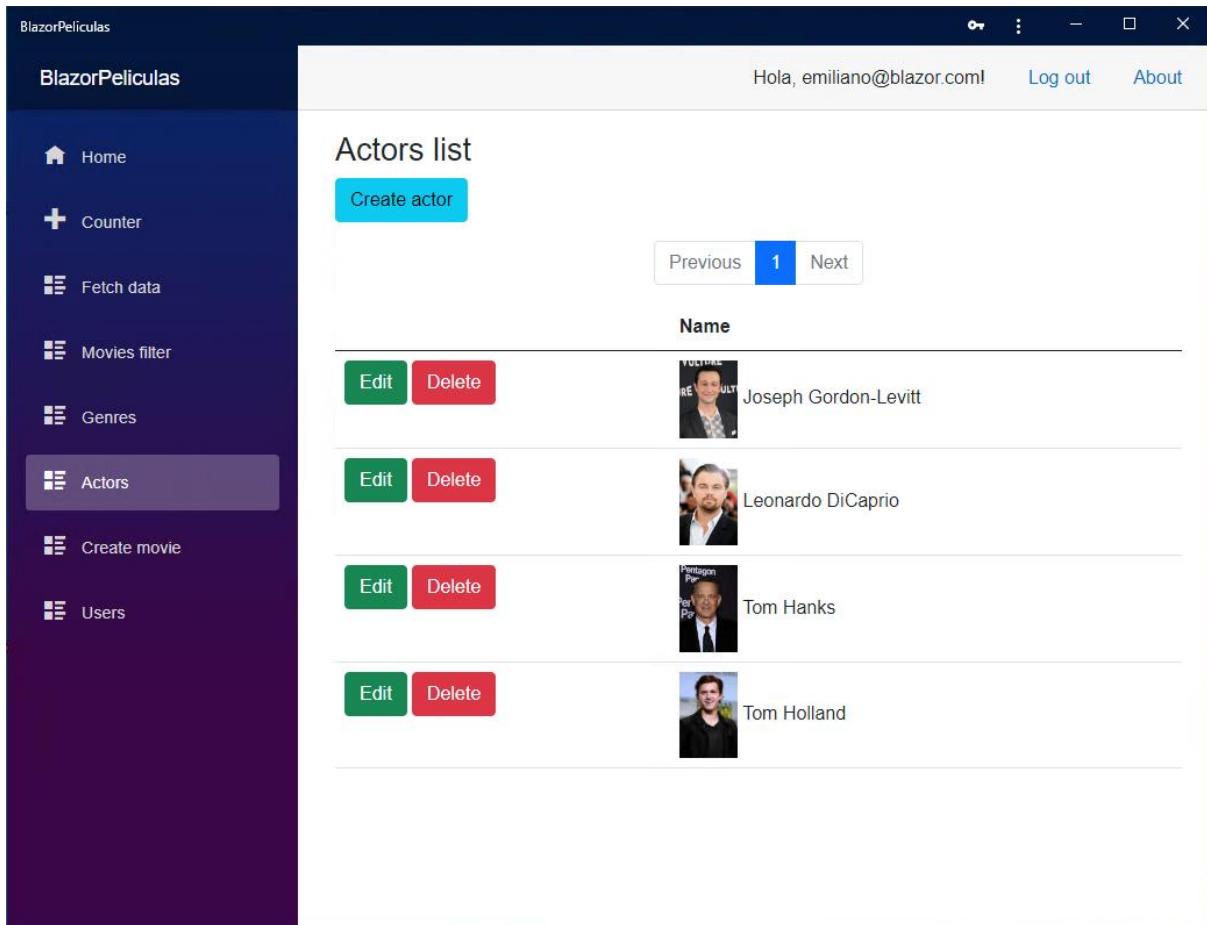
Probaremos la instalación de la PWA:



Vemos que nos crea un acceso directo en el escritorio y que se inicia la app sin ejecutar dentro del Chrome:



Incluso podemos loguearnos y navegar por la app:



The screenshot shows a Blazor application window titled "BlazorPelículas". On the left is a dark sidebar menu with the following items:

- Home
- Counter
- Fetch data
- Movies filter
- Genres
- Actors** (highlighted)
- Create movie
- Users

The main content area is titled "Actors list" and contains a table with four rows of actor data:

Name
 Joseph Gordon-Levitt
 Leonardo DiCaprio
 Tom Hanks
 Tom Holland

Each row has "Edit" and "Delete" buttons next to the actor's name.

Cacheando GETs

Tal como está la app, en modo offline no se puede navegar porque no se puede realizar los fetchs hacia el servidor. Lo que haremos es cachear los GETs para que si un usuario intenta hacer una petición que ya hizo anteriormente mientras está en modo offline, se muestre la información cacheada en vez de tirar error.

Trabajaremos, entonces, en nuestro SW. Precisamente en el evento `onFetch`. Lo primero que haremos es que si no es una petición GET retornaremos lo que el `fetch` del evento devuelva.

Creamos una variable `cacheNameDynamic` para guardar el nombre del caché en el que guardaremos contenido dinámico.

Primero abrimos el cache estático. Si lo tenemos cacheado respondemos con esa información directamente. Si no existe tal información, llamamos a `getAndUpdate` para que haga el `fetch`.

Primero intenta hacer el fetch (que sabemos que es un GET porque de no serlo, no habría llegado hasta allí ya que habría salido por el cuerpo del primer **IF** del **onFetch**). Si no lo puede hacer (porque estamos offline, por ejemplo) se va al catch e intenta obtener la info desde el cache dinámico y devolvemos lo que haya (o no) en el caché.

Si el fetch es exitoso, revisamos el content-type para no cachear contenido HTML. No queremos guardar HTML en el cache. Queremos guardar JSON, imágenes, etc.

Si no se trata de contenido HTML abrimos el **cacheNameDynamic** y guardamos el contenido que trajo el fetch (el **event.request** como key) en el caché. Es necesario clonar la respuesta porque usamos como response es un string sólo se puede leer una vez. Si no lo clonáramos luego de grabar en cache la respuesta estaría vacía.

service-worker.js

```
// Caution! Be sure you understand the caveats before publishing an application with
// offline support. See https://aka.ms/blazor-offline-considerations

self.importScripts('./service-worker-assets.js');
self.addEventListener('install', event => event.waitUntil(onInstall(event)));
self.addEventListener('activate', event => event.waitUntil(onActivate(event)));
self.addEventListener('fetch', event => event.respondWith(onFetch(event)));

const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const cacheNameDynamic = 'dynamic-cache';
const offlineAssetsInclude = [/\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/, /\.json$/, /\.css$/, /\.woff$/,
/\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/, /\.blat$/, /\.dat$/];
const offlineAssetsExclude = [/^service-worker\.js$/];

async function onInstall(event) {
    console.info('Service worker: Install');

    // Fetch and cache all matching items from the assets manifest
    const assetsRequests = self.assetsManifest.assets
        .filter(asset => offlineAssetsInclude.some(pattern => pattern.test(asset.url)))
        .filter(asset => !offlineAssetsExclude.some(pattern => pattern.test(asset.url)))
        .map(asset => new Request(asset.url, { integrity: asset.hash, cache: 'no-cache' }));
    await caches.open(cacheName).then(cache => cache.addAll(assetsRequests));
}

async function onActivate(event) {
    console.info('Service worker: Activate');

    // Delete unused caches
    const cacheKeys = await caches.keys();
    await Promise.all(cacheKeys
```

```
.filter(key => key.startsWith(cacheNamePrefix) && key !== cacheName)
.map(key => caches.delete(key)));
}

async function onFetch(event) {
  if (event.request.method !== 'GET') {
    return fetch(event.request);
  }

  let cachedResponse = null;
  // For all navigation requests, try to serve index.html from cache
  // If you need some URLs to be server-rendered, edit the following check to exclude those URLs
  const shouldServeIndexHtml = event.request.mode === 'navigate';

  const request = shouldServeIndexHtml ? 'index.html' : event.request;
  const cache = await caches.open(cacheName);
  cachedResponse = await cache.match(request);

  if(cachedResponse) {
    //Existe informacion estatica cacheada. La retornaremos.
    return cachedResponse;
  }

  var response = await getAndUpdate(event);

  return response;
}

async function getAndUpdate(event) {
  try {
    const response = await fetch(event.request);
    const contentType = response.headers.get('content-type');

    let saveInCache = true;
    if (contentType)
      saveInCache = !contentType.includes('text/html');

    if(saveInCache) {
      const cache = await caches.open(cacheNameDynamic);
      await cache.put(event.request, response.clone());
    }

    return response;
  }
  catch {
    //Si hay un error, no pudimos establecer la conexión
    const cache = await caches.open(cacheNameDynamic);
    return cache.match(event.request);
  }
}
```

Clean solution y Rebuild solution. Ejecutamos con F5 y nos aseguramos de que hay instalado y activado la última versión del SW:

<https://localhost:7152/> [Netv](#)

Source [service-worker.js](#)

Received 24/02/2023, 10:33:22
 Status ● #8 activated and is running [stop](#)

Clients <https://localhost:7152/> [focus](#)

Push [Test push message from DevTools.](#)

Navegamos a la película Spiderman. Abrimos **F12**, vamos a la solapa **Application** y buscamos la sección de cache (actualizamos el listado si es necesario). Vemos que ya hay un nuevo caché llamado **dynamic-cache** y que en él además de css, js e imágenes... ya está guardado el fetch del listado de películas como el de la info de la película 1 (Spiderman).

#	Name	Response-T...	Content-Type	Content-Le...	Time Cached	Vary Header
0	/ajax/libs/font-awesome/6.3.0/css/all.min.css	opaque	text/css; ch...	18,765	24/02/2023,...	Accept-Enc...
1	/ajax/libs/font-awesome/6.3.0/webfonts/fa-solid-900.woff2	cors	application/...	149,908	16/02/2023,...	Accept-Enc...
2	./framework/blazor-hotreload	basic	text/plain	0	24/02/2023,...	
3	./framework/blazor-hotreload.js	basic	application/...	743	24/02/2023,...	
4	/api/movies	basic	application/...	0	24/02/2023,...	
5	/api/movies/1	basic	application/...	0	24/02/2023,...	
6	/movies/61950841-92f2-44aa-a90d-135d6e353074.jpg	basic	image/jpeg	124,348	24/02/2023,...	
7	/movies/749598dc-384b-497f-984f-6ca1e8411327.jpg	basic	image/jpeg	419,890	24/02/2023,...	
8	/people/2b89d596-5f00-4ab3-9890-f2e853fbef96.jpg	basic	image/jpeg	12,094	24/02/2023,...	
9	/wikipedia/commons/b/b1>Loading_icon.gif?20151024034...	opaque	image/gif	17,490	27/01/2023,...	

Si abrimos el preview de </api/movies> vemos (corté los resúmenes para que no sean tan largos):

```
/api/movies
{
  "onBoard": [
    {
      "id": 1,
      "title": "Spiderman: Far from home",
      "summary": "### Summary\nIn Mexico, Nick Fury and Maria Hill investigate ....",
      "onBillboard": true,
      "trailer": "Nt9L1jCKGnE",
      "releaseDate": "2019-06-26T00:00:00",
      "poster": "movies/61950841-92f2-44aa-a90d-135d6e353074.jpg",
      "genresMovie": []
    }
  ]
}
```

```
"movieActor": [],
"votesMovies": [],
"trimmedTitle": "Spiderman: Far from home"
},
{
"id": 2,
"title": "Inception",
"summary": "### IMDb\nA thief who steals corporate secrets through the use of dream-sharing...",
"onBillboard": true,
"trailer": "YoHD9XEInc0",
"releaseDate": "2010-07-16T00:00:00",
"poster": "movies/749598dc-384b-497f-984f-6ca1e8411327.jpg",
"genresMovie": [],
"movieActor": [],
"votesMovies": [],
"trimmedTitle": "Inception"
}
],
"nextReleases": []
}
```

Y el de </api/movies/1> (corté los resúmenes y bios para que no sean tan largos):

```
/api/movies/1
{
  "movie": {
    "id": 1,
    "title": "Spiderman: Far from home",
    "summary": "### Summary\nIn Mexico, Nick Fury and Maria Hill investigate ...",
    "onBillboard": true,
    "trailer": "Nt9L1jCKGnE",
    "releaseDate": "2019-06-26T00:00:00",
    "poster": "movies/61950841-92f2-44aa-a90d-135d6e353074.jpg",
    "genresMovie": [
      {
        "movielID": 1,
        "genreID": 2,
        "genre": {
          "id": 2,
          "name": "Action",
          "genres": [
            null
          ]
        }
      },
      "movie": null
    ],
    "movielID": 1,
  }
}
```

```
"genreID": 3,
"genre": {
    "id": 3,
    "name": "Adventure",
    "genres": [
        null
    ]
},
"movie": null
},
{
    "movielD": 1,
    "genreID": 4,
    "genre": {
        "id": 4,
        "name": "Sci-Fi",
        "genres": [
            null
        ]
    },
    "movie": null
}
],
"movieActor": [
{
    "actorID": 1,
    "movielD": 1,
    "actor": {
        "id": 1,
        "name": "Tom Holland",
        "bio": "#### Thomas Stanley Holland\nHe was born 1 June 1996. He is an English actor...",
        "photo": "people/2b89d596-5f00-4ab3-9890-f2e853fbef96.jpg",
        "birthDate": "1996-06-01T00:00:00",
        "character": null,
        "movieActor": [
            null
        ]
    },
    "movie": null,
    "character": "Peter Parker / Spiderman",
    "orden": 1
}
],
"votesMovies": [],
"trimmedTitle": "Spiderman: Far from home"
},
"genres": [
{
    "id": 2,
    "name": "Action",
```

```
"genres": [
  {
    "movieID": 1,
    "genreID": 2,
    "genre": null,
    "movie": {
      "id": 1,
      "title": "Spiderman: Far from home",
      "summary": "### Summary\nIn Mexico, Nick Fury and Maria Hill investigate ...",
      "onBillboard": true,
      "trailer": "Nt9L1jCKGnE",
      "releaseDate": "2019-06-26T00:00:00",
      "poster": "movies/61950841-92f2-44aa-a90d-135d6e353074.jpg",
      "genresMovie": [
        null,
        {
          "movieID": 1,
          "genreID": 3,
          "genre": {
            "id": 3,
            "name": "Adventure",
            "genres": [
              null
            ],
            "movie": null
          },
          {
            "movieID": 1,
            "genreID": 4,
            "genre": {
              "id": 4,
              "name": "Sci-Fi",
              "genres": [
                null
              ],
              "movie": null
            }
          }
        ],
        "movieActor": [
          {
            "actorID": 1,
            "movieID": 1,
            "actor": {
              "id": 1,
              "name": "Tom Holland",
              "bio": "### Thomas Stanley Holland\nHe was born 1 June 1996. He is an English
actor...",
              "photo": "people/2b89d596-5f00-4ab3-9890-f2e853fbef96.jpg"
            }
          }
        ]
      }
    }
  }
]
```

```
"birthDate": "1996-06-01T00:00:00",
"character": null,
"movieActor": [
    null
],
"movie": null,
"character": "Peter Parker / Spiderman",
"orden": 1
},
],
"votesMovies": [],
"trimmedTitle": "Spiderman: Far from home"
}
]
},
{
"id": 3,
"name": "Adventure",
"genres": [
{
"movielID": 1,
"genreID": 3,
"genre": null,
"movie": {
"id": 1,
"title": "Spiderman: Far from home",
"summary": "### Summary\nIn Mexico, Nick Fury and Maria Hill investigate ...",
"onBillboard": true,
"trailer": "Nt9L1jCKGnE",
"releaseDate": "2019-06-26T00:00:00",
"poster": "movies/61950841-92f2-44aa-a90d-135d6e353074.jpg",
"genresMovie": [
{
"movielID": 1,
"genreID": 2,
"genre": {
"id": 2,
"name": "Action",
"genres": [
null
]
},
"movie": null
},
null,
{
"movielID": 1,
"genreID": 4,
```

```
"genre": {
    "id": 4,
    "name": "Sci-Fi",
    "genres": [
        null
    ],
    "movie": null
},
],
"movieActor": [
    {
        "actorID": 1,
        "movielD": 1,
        "actor": {
            "id": 1,
            "name": "Tom Holland",
            "bio": "### Thomas Stanley Holland\nHe was born 1 June 1996. He is an English
actor...",
            "photo": "people/2b89d596-5f00-4ab3-9890-f2e853fbebe96.jpg",
            "birthDate": "1996-06-01T00:00:00",
            "character": null,
            "movieActor": [
                null
            ],
            "movie": null,
            "character": "Peter Parker / Spiderman",
            "orden": 1
        }
    },
    "votesMovies": [],
    "trimmedTitle": "Spiderman: Far from home"
},
]
},
{
    "id": 4,
    "name": "Sci-Fi",
    "genres": [
        {
            "movielD": 1,
            "genreID": 4,
            "genre": null,
            "movie": {
                "id": 1,
                "title": "Spiderman: Far from home",
                "summary": "### Summary\nIn Lxtenco, Mexico, Nick Fury and Maria Hill investigate ...",
                "onBillboard": true,
            }
        }
    ]
}
```

```

        "trailer": "Nt9L1jCKGnE",
        "releaseDate": "2019-06-26T00:00:00",
        "poster": "movies/61950841-92f2-44aa-a90d-135d6e353074.jpg",
        "genresMovie": [
            {
                "movielID": 1,
                "genreID": 2,
                "genre": {
                    "id": 2,
                    "name": "Action",
                    "genres": [
                        null
                    ]
                },
                "movie": null
            },
            {
                "movielID": 1,
                "genreID": 3,
                "genre": {
                    "id": 3,
                    "name": "Adventure",
                    "genres": [
                        null
                    ]
                },
                "movie": null
            },
            null
        ],
        "movieActor": [
            {
                "actorID": 1,
                "movielID": 1,
                "actor": {
                    "id": 1,
                    "name": "Tom Holland",
                    "bio": "### Thomas Stanley Holland\nHe was born 1 June 1996. He is an English
actor...",
                    "photo": "people/2b89d596-5f00-4ab3-9890-f2e853fbef96.jpg",
                    "birthDate": "1996-06-01T00:00:00",
                    "character": null,
                    "movieActor": [
                        null
                    ]
                },
                "movie": null,
                "character": "Peter Parker / Spiderman",
                "orden": 1
            }
        ]
    }
}

```

```
        ],
        "votesMovies": [],
        "trimmedTitle": "Spiderman: Far from home"
    }
}
],
"actors": [
{
    "id": 1,
    "name": "Tom Holland",
    "bio": null,
    "photo": "people/2b89d596-5f00-4ab3-9890-f2e853fbbe96.jpg",
    "birthDate": null,
    "character": "Peter Parker / Spiderman",
    "movieActor": []
},
],
"userVote": 0,
"votesMedia": 4
}
```

Si nos vamos a modo offline, vemos que el tanto el listado como la info de la película Spiderman funcionan correctamente:

localhost:7152

BlazorPelículas

- Home
- Counter
- Fetch data
- Movies filter

On billboard

Network

```

dotnet.7.0.3.uxh8y7lsj4.js
  ...
  function mono_wasm_runtime_ready() {
    ...
  }
  function mono_wasm_fire_debugger_agent_message() {
    ...
  }
  ...
  
```

Line 3, Column 6655

Coverage: n/a

localhost:7152/movie/1/Spiderman:-Far-from-home

BlazorPelículas

- Home
- Counter
- Fetch data
- Movies filter

Spiderman: Far from home (2019)

Action Adventure Sci-Fi | 26 06 2019 | Media: 4/5 | Your vote: ★★★★★

Network

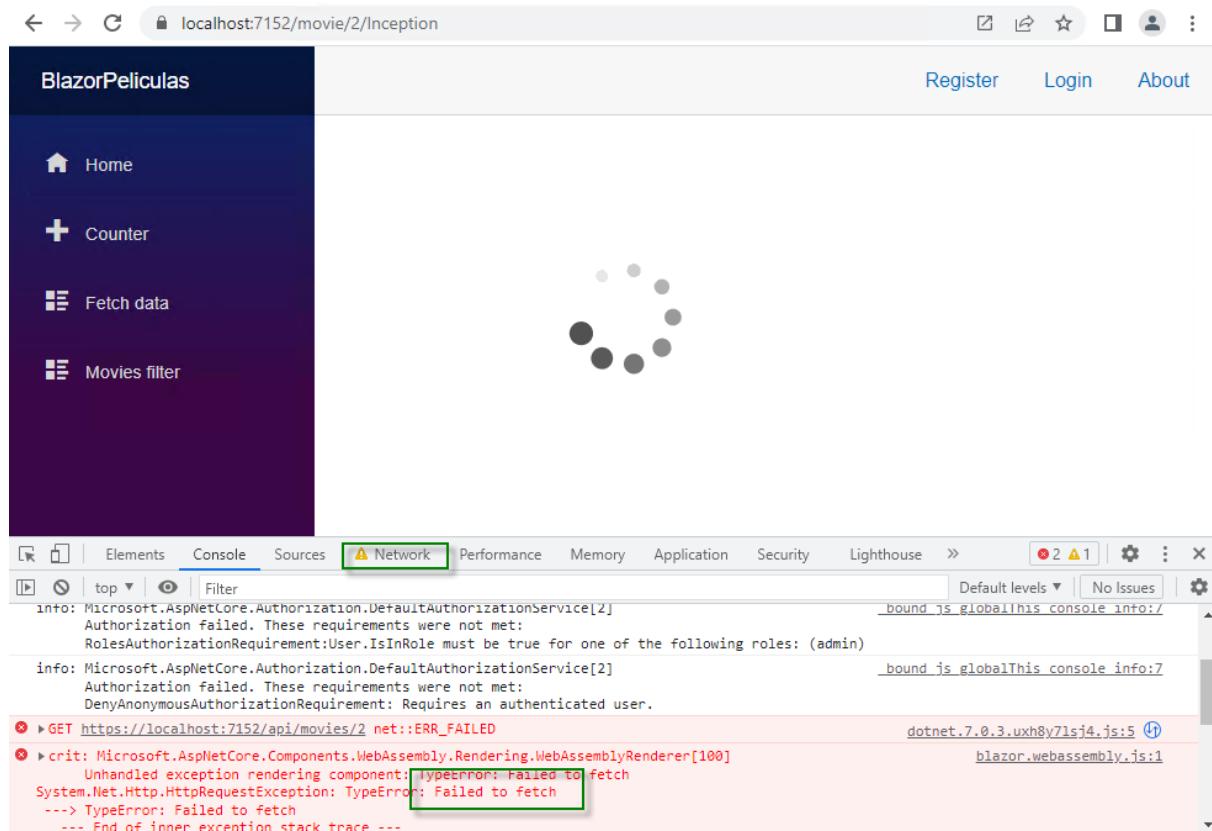
```

dotnet.7.0.3.uxh8y7lsj4.js
  ...
  function mono_wasm_runtime_ready() {
    ...
  }
  function mono_wasm_fire_debugger_agent_message() {
    ...
  }
  ...
  
```

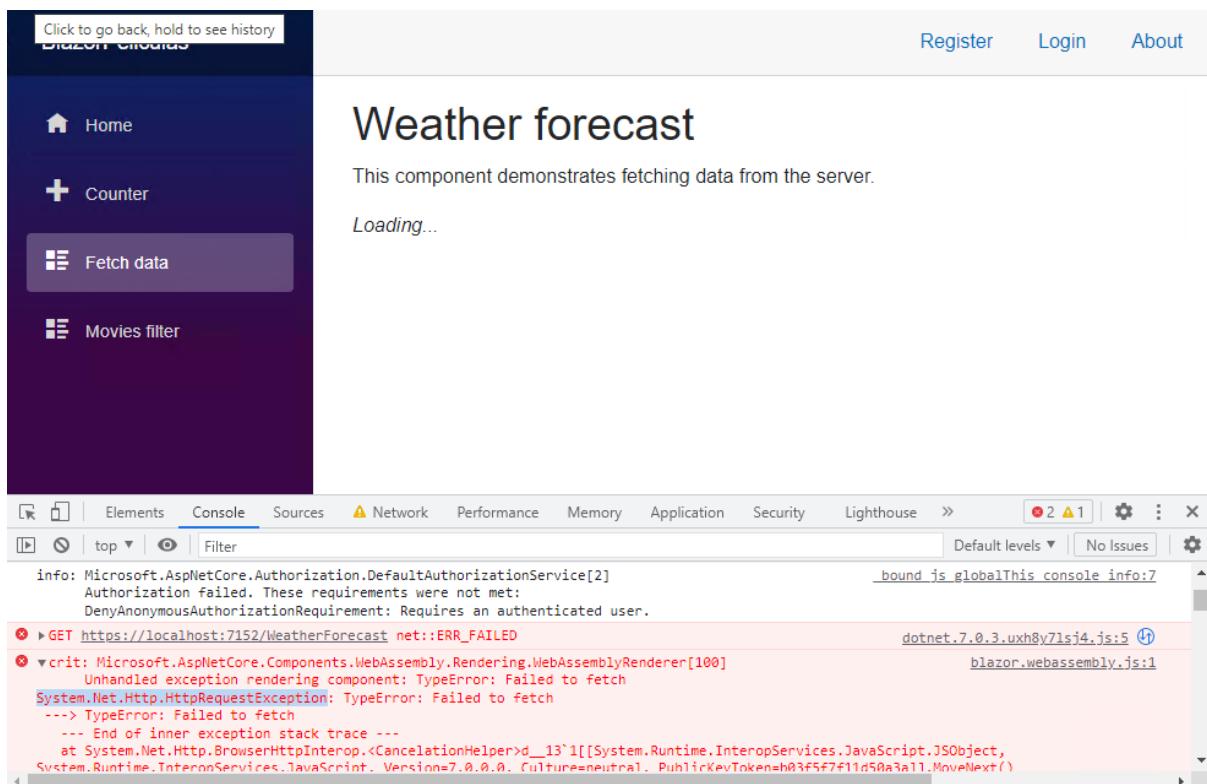
Line 3, Column 6655

Coverage: n/a

Pero si intentamos ir a una página que no hayamos visitado (y no está cacheada), fallará el fetch correspondiente:



Lo mismo si intentáramos ir a Fetch Data:



Lo ideal para que el usuario no se encuentre con estos errores feos es hacer un try catch. En la imagen anterior se ve que el error es **System.Net.Http.HttpRequestException**. Agregaremos un try/catch en el componente de FetchData.

FetchData.razor

```

@page "/fetchdata"
@using BlazorPeliculas.Shared
@inject HttpClient Http

<PageTitle>Weather forecast</PageTitle>

<h1>Weather forecast</h1>
<h2>@message</h2>

<p>This component demonstrates fetching data from the server.</p>

@if (forecasts == null)
{
    <p><em>Loading...</em></p>
}
else
{
    <table class="table">
        <thead>
            <tr>
  
```

```
<th>Date</th>
<th>Temp. (C)</th>
<th>Temp. (F)</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
@foreach (var forecast in forecasts)
{
    <tr>
        <td>@forecast.Date.ToShortDateString()</td>
        <td>@forecast.TemperatureC</td>
        <td>@forecast.TemperatureF</td>
        <td>@forecast.Summary</td>
    </tr>
}
</tbody>
</table>
}

@code {
private WeatherForecast[]? forecasts;
string message = string.Empty;

protected override async Task OnInitializedAsync()
{
    try {
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("WeatherForecast");
    }
    catch(Exception ex) {
        if(ex.GetType().ToString() == "System.Net.Http.HttpRequestException") {
            //No se pudo comunicar con el servidor.
            message = "A connection error to the server has occurred.";
        }
    }
}
}
```

El primer intento falla porque está trayendo información cacheada. Esto es uno de los problemas de utilizar modo offline en desarrollo. Vamos a **Application > Service workers** y elegimos **Update on reload**.

The screenshot shows a Blazor application titled "BlazorPelículas". On the left is a sidebar with navigation links: Home, Counter, Fetch data (which is highlighted), and Movies filter. The main content area displays a heading "Weather forecast" followed by an error message in a red-bordered box: "A connection error to the server has occurred.". Below the error message, it says "This component demonstrates fetching data from the server." and "Loading...". At the bottom of the main content, there is some raw HTML code. To the right of the main content is a developer tools window with the "Network" tab selected. The Network tab shows a failed request with the status code 0. The "Elements" tab shows the DOM structure. The "Styles" tab shows the CSS rules applied to the elements.

IndexedDB

Veremos como guardar información localmente para poder borrar, modificar, etc de manera offline. De esta forma, una vez que volvamos a tener conexión se puedan aplicar todos las peticiones que no pudieron mandarse oportunamente.

No usaremos IndexedDB directamente sino una librería que nos hará la vida más sencilla: <https://dexie.org/>

Agregamos la librería al index.html

```
index.html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no" />
    <title>BlazorPelículas</title>
    <base href="/" />
    <link href="css/bootstrap/bootstrap.min.css" rel="stylesheet" />
    <link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.3.0/css/all.min.css" rel="stylesheet" />
    <link href="css/app.css" rel="stylesheet" />
```

```
<link rel="icon" type="image/png" href="favicon.png" />
<link href="BlazorPeliculas.Client.styles.css" rel="stylesheet" />
<link href="_content/Blazored.Typeahead/blazored-typeahead.css" rel="stylesheet" />
<link href="manifest.json" rel="manifest" />
<link rel="apple-touch-icon" sizes="512x512" href="icon-512.png" />
<link rel="apple-touch-icon" sizes="192x192" href="icon-192.png" />
</head>

<body>
<div id="app">
<svg class="loading-progress">
<circle r="40%" cx="50%" cy="50%" />
<circle r="40%" cx="50%" cy="50%" />
</svg>
<div class="loading-progress-text"></div>
</div>

<div id="blazor-error-ui">
An unhandled error has occurred.
<a href="" class="reload">Reload</a>
<a class="dismiss">X</a>
</div>
<script src="_framework/blazor.webassembly.js"></script>
<script>navigator.serviceWorker.register('service-worker.js');</script>
<script src="https://unpkg.com/dexie@3.2.3/dist/dexie.js"></script>
<script src="js/Utilities.js"></script>
<script src="_content/CurrieTechnologies.Razor.Sweetalert2/sweetalert2.min.js"></script>
<script src="_content/Blazored.Typeahead/blazored-typeahead.js"></script>
</body>

</html>
```

Lo primero que haremos es crear nuestra base de IndexedDB. Podemos crear un nuevo archivo js pero nosotros utilizaremos el mismo [js/Utilities.js](#). Es importante que el js de Dexie esté antes que el de Utilities.js para que el IntelliSense sepa interpretar los objetos de esa librería.

Primero creamos la db y le asignamos un número de versión. En ella crearemos 2 tablas. En la primera (la llamaremos [create](#)) guardaremos las entidades que el usuario intentó crear (géneros, películas, etc); en la segunda (la llamaremos [delete](#)) las entidades que intentó borrar. El [++](#) es para indicar que el campo será autoincrementable.

Estas bases de datos tienen la peculiaridad de que son de esquema libre en lo que se refiere a sus columnas y por tanto, solamente indicaremos lo que va a ser como la llave primaria, entre comillas de mi store o tabla.

Luego cuando vayamos a crear o insertar un registro en esas tablas, vamos a poder hacerlo con la estructura que deseemos.

Por eso aquí solamente estamos indicando lo que viene siendo nuestro campo de id.

Utilities.js

```
function DotNetStaticTest() {
    DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
        .then(result => {
            console.log('Count from js: ' + result);
        });
}

function DotNetInstanceTest(dotNetHelper) {
    dotNetHelper.invokeMethodAsync("IncrementCount");
}

function inactiveTimer(dotnetHelper) {
    var timer;

    document.onmousemove = resetimer;
    document.onkeypress = resetimer;

    function resetimer() {
        clearTimeout(timer);
        timer = setTimeout(logout, 5 * 60 * 1000); // 5 minutos
    }

    function logout() {
        dotnetHelper.invokeMethodAsync("Logout");
    }
}

var db = new Dexie("mydb");
var dbVersion = 1;

db.version(dbVersion).stores({
    create: 'id++',
    delete: 'id++'
});
```

Con ese sencillo código se creará la BD con las 2 tablas. Agregaremos la siguiente línea al final para poder ver la tabla con info:

```
db['create'].put({ url: 'api/genres', body: { name: 'Comedy' } });
```

Ejecutamos y presionamos **Ctrl+F5** en Chrome para que se cargue el nuevo **Utilities.js**. Vemos que tenemos nuestra base mydb con 2 tablas y que en la tabla create se encuentra el registro de prueba que recién agregamos.

#	Key (Key path: "id")	Value
0	1	{url: 'api/genres', body: {...}, id: 1} ↳ body: {name: 'Comedy'} ↳ id: 1 ↳ url: "api/genres"

Presionamos delete sobre el registro para eliminarlo y sacamos el código de prueba para que no se vuelva a crear.

Creamos el componente **Synchronizer.razor**.

Para poder traer todos los registros pendientes, crearemos una función asíncrona en **Utilities.js**. Básicamente devolverá un objeto con 2 arrays: uno con los elementos a crear y el otro con los elementos a borrar. Otro método para borrar un registro. Y un tercer método para obtener la cantidad de registros pendientes.

Utilities.js

```

function DotNetStaticTest() {
  DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
    .then(result => {
      console.log('Count from js: ' + result);
    });
}

function DotNetInstanceTest(dotNetHelper) {
  dotNetHelper.invokeMethodAsync("IncrementCount");
}

function inactiveTimer(dotnetHelper) {
  var timer;

  document.onmousemove = resetimer;
  document.onkeypress = resetimer;

  function resetimer() {
    clearTimeout(timer);
    timer = setTimeout(logout, 5 * 60 * 1000); // 5 minutos
  }

  function logout() {
    dotnetHelper.invokeMethodAsync("Logout");
  }
}

```

```
}
```

```
var db = new Dexie("mydb");
var dbVersion = 1;
```

```
db.version(dbVersion).stores({
  create: 'id++',
  delete: 'id++'
});
```

```
async function getPendingRecords() {
  return await [
    ObjectsToCreate: await db.create.toArray(),
    ObjectsToDelete: await db.delete.toArray()
  ];
}
```

```
async function deleteRecord(table, id) {
  await db[table].where({ "id": id }).delete();
}
```

```
async function getPendingRecordCount () {
  var count = await db.create.count();
  count += await db.delete.count();

  return count;
}
```

Esta función debe devolver una clase con todos los datos necesarios. Creamos la clase **DbLocalRecord.cs** en la carpeta **Helpers**.

DbLocalRecord.cs

```
using System.Text.Json;
```

```
namespace BlazorPeliculas.Client.Helpers {
  public class DbLocalRecord {
    public int Id { get; set; }
    public string Url { get; set; }
    public JsonElement Body { get; set; }
  }

  public class DbLocalRecords {
    public List<DbLocalRecord> ObjectsToCreate { get; set; } = new List<DbLocalRecord>();
    public List<DbLocalRecord> ObjectsToDelete { get; set; } = new List<DbLocalRecord>();

    //int GetPending => ObjectsToCreate.Count + ObjectsToDelete.Count;

    public int GetPending() {
      var count = 0;
```

```
        count += ObjectsToCreate.Count;
        count += ObjectsToDelete.Count;

        return count;
    }
}
```

Agregamos los métodos para poder ser invocados desde JS.

IJSRuntimeExtensionMethods.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.JSInterop;
using System.Runtime.CompilerServices;

namespace BlazorPeliculas.Client.Helpers {
    public static class IJSRuntimeExtensionMethods {

        public static async ValueTask<DbLocalRecords> GetPendingRecords(this IJSRuntime js) {
            return await js.InvokeAsync<DbLocalRecords>("getPendingRecords");
        }

        public static async ValueTask deleteRecord(this IJSRuntime js, string table, int id) {
            await js.InvokeVoidAsync("deleteRecord", table, id);
        }

        public static async ValueTask<int> GetPendingRecordCount(this IJSRuntime js) {
            return await js.InvokeAsync<int>("getPendingRecordCount");
        }

        public static async ValueTask<bool> Confirm(this IJSRuntime js, string message) {
            await js.InvokeVoidAsync("console.log", $"Asking: «{message}»");
            return await js.InvokeAsync<bool>("confirm", message);
        }
        public static ValueTask<object> SetInLocalStorage(this IJSRuntime js,
            string key, string value) {
            return js.InvokeAsync<object>("localStorage.setItem", key, value);
        }

        public static ValueTask<object> GetFromLocalStorage(this IJSRuntime js,
            string key) {
            return js.InvokeAsync<object>("localStorage.getItem", key);
        }

        public static ValueTask<object> RemoveFromLocalStorage(this IJSRuntime js,
            string key) {
            return js.InvokeAsync<object>("localStorage.removeItem", key);
        }
    }
}
```

El bloque de **try/catch** es para evitar recibir un error si el usuario intenta sincronizar cuando no tiene internet. El **EnsureSuccessStatusCode** es para asegurarnos que no haya tirado error. De lo contrario, que se vaya al catch.

Synchronizer.razor

```
@inject IJSRuntime js
@inject IRepository repository

<div style="color:white; margin-bottom:1em; border-top:1px solid white;">
    <div style="margin-left:5px;margin-top:5px;">
        @if(synchronizing) {
            <div>Synchronizing...</div>
        }
        else if(hasError) {
            <div>Synchronizing failed.</div>
            <button @onclick="SinchronizeClick">Re-try</button>
        }
        else {
            <div>Pending synchronization: @pending </div>
            @if(pending > 0) {
                <button @onclick="SinchronizeClick">Syncrhonize</button>
            }
        }
    </div>
</div>

@code {
    int pending = 0;
    bool synchronizing = false;
    bool hasError = false;

    protected override async Task OnInitializedAsync() {
        await Sinchronize();
    }

    private async Task SinchronizeClick() {
        await Sinchronize();
    }

    public async Task updatePending() {
        pending = await js.GetPendingRecordCount();
        StateHasChanged();
    }

    private async Task Sinchronize() {
        var dbLocalRecords = await js.GetPendingRecords();

        pending = dbLocalRecords.GetPending();
```

```
if(pending == 0) return; //No debería pasar nunca, si no hubiera, el botón no se vería.

synchronizing = true;
StateHasChanged();

try {
    foreach(var entity in dbLocalRecords.ObjectsToCreate) {
        var response = await repository.Post(entity.Url, entity.Body);
        response.httpResponseMessage.EnsureSuccessStatusCode();

        await js.deleteRecord("createStore", entity.Id);
    }

    foreach(var entity in dbLocalRecords.ObjectsToDelete) {
        var response = await repository.Delete(entity.Url);
        response.httpResponseMessage.EnsureSuccessStatusCode();

        await js.deleteRecord("deleteStore", entity.Id);
    }

    hasError = false;
}
catch(Exception ex) {
    hasError = true;
}

synchronizing = false;
StateHasChanged();
}
}
```

Agregamos el nuevo componente en MainLayout.razor.

MainLayout.razor

```
@inherits LayoutComponentBase
@inject BlazorPeliculas.Client.Auth.TokenRenewer tokenRenewer
@inject IJSRuntime js
@inject NavigationManager navManager

<div class="page">
    <div class="sidebar" style="display:flex;flex-direction:column;justify-content:space-between">
        <NavMenu />
    </div>

    <main>
        <div class="top-row px-4">
            <AuthLinks />
            <a href="https://docs.microsoft.com/aspnet/" target="_blank">About</a>
        </div>
    </main>
</div>
```

```
</div>

<article class="content px-4">
    @Body
</article>
</main>
</div>

@code {
    [CascadingParameter]
    public Task<AuthenticationState> AuthenticationStateTask { get; set; } = null;

    protected async override Task OnInitializedAsync() {
        await js.InvokeVoidAsync("inactiveTimer", DotNetObjectReference.Create(this));
        tokenRenewer.Start();
    }

    [JSInvokable]
    public async Task Logout() {
        var authState = await AuthenticationStateTask;
        if(authState.User.Identity!.IsAuthenticated)
            navManager.NavigateTo("/logout");
    }
}
```

NavMenu.razor

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">BlazorPelículas</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="oi oi-home" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="oi oi-plus" aria-hidden="true"></span> Counter
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="fetchdata">
                <span class="oi oi-database" aria-hidden="true"></span> Fetch Data
            </NavLink>
        </div>
    </nav>
</div>
```

```
<span class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
</NavLink>
</div>

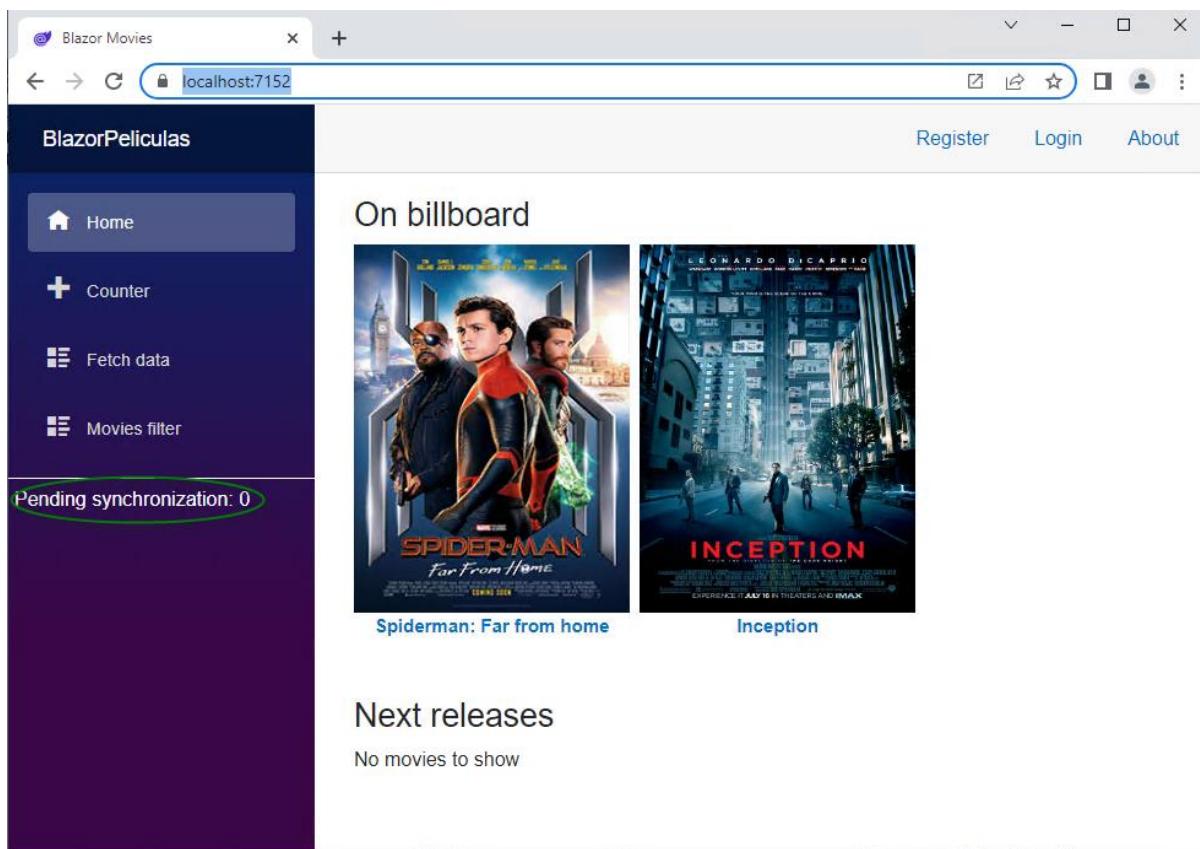
<div class="nav-item px-3">
    <NavLink class="nav-link" href="movies/filter">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Movies filter
    </NavLink>
</div>

<AuthorizeView Roles="admin">
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="genres">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Genres
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="actors">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Actors
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="movies/create">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Create movie
        </NavLink>
    </div>
    <div class="nav-item px-3">
        <NavLink class="nav-link" href="users">
            <span class="oi oi-list-rich" aria-hidden="true"></span> Users
        </NavLink>
    </div>
    </AuthorizeView>
</nav>
<Synchronizer />
</div>

@code {
    private bool collapseNavMenu = true;

    private string? NavMenuCssClass => collapseNavMenu ? "collapse" : null;

    private void ToggleNavMenu()
    {
        collapseNavMenu = !collapseNavMenu;
    }
}
```



Grabando géneros offline

Crearemos un método en Utilities.js para poder grabar un registro cuando no podamos grabar/borrar por estar offline.

Utilities.js

```

function DotNetStaticTest() {
    DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
        .then(result => {
            console.log('Count from js: ' + result);
        });
}

function DotNetInstanceTest(dotNetHelper) {
    dotNetHelper.invokeMethodAsync("IncrementCount");
}

function inactiveTimer(dotnetHelper) {
    var timer;

    document.onmousemove = resetimer;
    document.onkeypress = resetimer;

    function resetimer() {
        clearTimeout(timer);
    }
}

```

```
        timer = setTimeout(logout, 5 * 60 * 1000); // 5 minutos
    }

    function logout() {
        dotnetHelper.invokeMethodAsync("Logout");
    }
}

var db = new Dexie("mydb");
var dbVersion = 1;

db.version(dbVersion).stores({
    createStore: 'id++',
    deleteStore: 'id++'
});

async function getPendingRecords() {
    return await {
        ObjectsToCreate: await db.createStore.toArray(),
        ObjectsToDelete: await db.deleteStore.toArray()
    };
}

async function deleteRecord(table, id) {
    await db[table].where({ "id": id }).delete();
}

async function getPendingRecordCount() {
    var count = await db.createStore.count();
    count += await db.deleteStore.count();

    return count;
}

async function saveCreateRecord (url, body) {
    await db.createStore.put({ url, body: JSON.parse(body) });
}

async function saveDeleteRecord (url) {
    await db.deleteStore.put({ url });
}
```

IJSRuntimeExtensionMethods.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.JSInterop;
using System.Runtime.CompilerServices;
using System.Text.Json;

namespace BlazorPeliculas.Client.Helpers {
```

```

public static class IJSRuntimeExtensionMethods {

    public static async ValueTask<DbLocalRecords> GetPendingRecords(this IJSRuntime js) {
        return await js.InvokeAsync<DbLocalRecords>("getPendingRecords");
    }

    public static async ValueTask deleteRecord(this IJSRuntime js, string table, int id) {
        await js.InvokeVoidAsync("deleteRecord", table, id);
    }

    public static async ValueTask saveCreateRecord<T>(this IJSRuntime js, string url, T entity) {
        var body = JsonSerializer.Serialize(entity);
        await js.InvokeVoidAsync("saveCreateRecord", url, body);
    }

    public static async ValueTask saveDeleteRecord(this IJSRuntime js, string url) {
        await js.InvokeVoidAsync("saveDeleteRecord", url);
    }

    public static async ValueTask<int> GetPendingRecordCount(this IJSRuntime js) {
        return await js.InvokeAsync<int>("getPendingRecordCount");
    }

    public static async ValueTask<bool> Confirm(this IJSRuntime js, string message) {
        await js.InvokeVoidAsync("console.log", $"Asking: «{message}»");
        return await js.InvokeAsync<bool>("confirm", message);
    }

    public static ValueTask<object> SetInLocalStorage(this IJSRuntime js,
        string key, string value) {
        return js.InvokeAsync<object>("localStorage.setItem", key, value);
    }

    public static ValueTask<object> GetFromLocalStorage(this IJSRuntime js,
        string key) {
        return js.InvokeAsync<object>("localStorage.getItem", key);
    }

    public static ValueTask<object> RemoveFromLocalStorage(this IJSRuntime js,
        string key) {
        return js.InvokeAsync<object>("localStorage.removeItem", key);
    }
}

```

CreateGenre.razor

```

@page "/genres/create"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swal

```

```
@attribute [Authorize(Roles = "admin")]
@inject IJSRuntime js

<h3>Create Genre</h3>

<GenreForm @ref="genreForm" Genre="genre" OnValidSubmit="Create" />

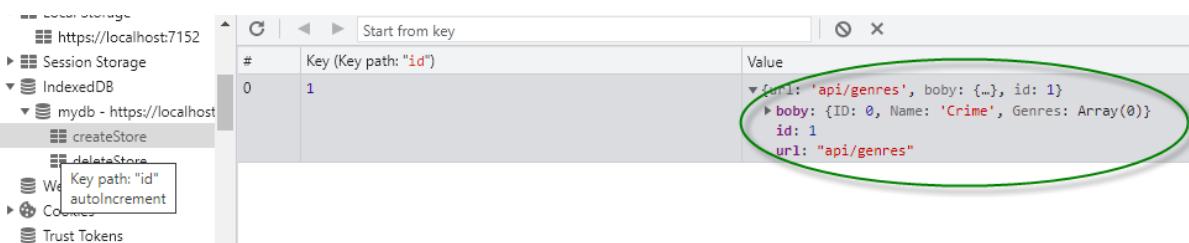
@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío
    private GenreForm? genreForm;

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private async Task Create() {
        try {
            var httpResponse = await repository.Post("api/genres", genre);

            if (httpResponse.Error) {
                var ErrMessage = await httpResponse.GetErrorMessage();
                await swal.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
            else {
                genreForm!.formPostedCorrectly = true;
                navManager.NavigateTo("/genres");
            }
        }
        catch(Exception ex) {
            if(ex.GetType().ToString() == "System.Net.Http.HttpRequestException") {
                await js.saveCreateRecord("api/genres", genre);
                genreForm!.formPostedCorrectly = true;
                navManager.NavigateTo("/genres");
            }
            else
                throw ex;
        }
    }
}
```

Para probar, entramos en el F12 de Chrome y vamos a la solapa de Network para asegurarnos de tener el **Disable cache** desactivado porque necesitamos caché. También revisamos que el SW esté instalado y activado.

Intentamos crear el género Crime en modo offline:



Una vez que se recupera la conexión, el componente Synchronizer debería hacer el POST correspondiente. Sin embargo, me tira un error (entity viene en null) y no actualiza.

De todas formas, cuando se graba un género de manera offline, el listado de género no se actualiza (es coherente porque lo trae del caché).

Evento del Synchronizer

Crearemos un evento para que el componente esté escuchando y poder enterarse cuando alguien modificó el listado de pendientes para así poder actualizar el contador.

Crearmos una clase **PendingState.cs** en la carpeta **Helpers**.

PendingState.cs

```
namespace BlazorPeliculas.Client.Helpers {
    public class PendingState {
        public event Func<Task> OnUpdatePendingSynchronizations;
        public async Task NotifyUpdatePendingSynchronizations() => await
OnUpdatePendingSynchronizations?.Invoke();
    }
}
```

Agregamos un **Singleton** para usarlo en **Program.cs**:

Program.cs

```
using BlazorPeliculas.Client;
using BlazorPeliculas.Client.Auth;
using BlazorPeliculas.Client.Helpers;
using BlazorPeliculas.Client.Repositories;
using CurrieTechnologies.Razor.SweetAlert2;
using Microsoft.AspNetCore.Components.Authorization;
using Microsoft.AspNetCore.Components.Web;
using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
using Microsoft.Extensions.DependencyInjection;

var builder = WebAssemblyHostBuilder.CreateDefault(args);
builder.RootComponents.Add<App>("#app");
builder.RootComponents.Add<HeadOutlet>("head::after");

builder.Services.AddSingleton(sp => new HttpClient { BaseAddress = new
```

```
Uri(builder.HostEnvironment.BaseAddress));
configureServices(builder.Services);
await builder.Build().RunAsync();

void configureServices(IServiceCollection services) {
    services.AddScoped< IRepository, Repository>(); /* Inyectar un IRepository.
    Sin embargo, en tiempo de ejecución lo que se va a hacer es que se va a proveer una instancia de la
    clase Repository. */
    services.AddSweetAlert2();
    services.AddAuthorizationCore();
    services.AddSingleton<PendingState>();

    services.AddScoped<JWTAuthProvider>();
    services.AddScoped<AuthenticationStateProvider, JWTAuthProvider>(provider =>
        provider.GetRequiredService<JWTAuthProvider>());

    services.AddScoped<ILoginService, JWTAuthProvider>(provider =>
        provider.GetRequiredService<JWTAuthProvider>());

    services.AddScoped<TokenRenewer>();
}
```

Inyectamos **PendingState** y nos suscribimos al evento en el componente **Synchronizer.razor**.

Synchronizer.razor

```
@inject IJSRuntime js
@inject IRepository repository
@inject PendingState pendingState

<div style="color:white; margin-bottom:1em; border-top:1px solid white;">
    <div style="margin-left:5px;margin-top:5px;">
        @if(synchronizing) {
            <div>Synchronizing...</div>
        }
        else if(hasError) {
            <div>Synchronizing failed.</div>
            <button @onclick="SinchronizeClick">Re-try</button>
        }
        else {
            <div>Pending synchronization: @pending </div>
            @if(pending > 0) {
                <button @onclick="SinchronizeClick">Syncrhonize</button>
            }
        }
    </div>
</div>
```

```
@code {
    int pending = 0;
    bool synchronizing = false;
    bool hasError = false;

    protected override async Task OnInitializedAsync() {
        pendingState.OnUpdatePendingSynchronizations += updatePending;
        await Sinchronize();
    }

    private async Task SinchronizeClick() {
        await Sinchronize();
    }

    public async Task updatePending() {
        pending = await js.GetPendingRecordCount();
        StateHasChanged();
    }

    private async Task Sinchronize() {
        var dbLocalRecords = await js.GetPendingRecords();

        pending = dbLocalRecords.GetPending();
        if(pending == 0) return; //No debería pasar nunca, si no hubiera, el botón no se vería.

        synchronizing = true;
        StateHasChanged();

        try {
            foreach(var entity in dbLocalRecords.ObjectsToCreate) {
                var response = await repository.Post(entity.Url, entity.Body);
                response.httpResponseMessage.EnsureSuccessStatusCode();

                await js.deleteRecord("createStore", entity.Id);
            }

            foreach(var entity in dbLocalRecords.ObjectsToDelete) {
                var response = await repository.Delete(entity.Url);
                response.httpResponseMessage.EnsureSuccessStatusCode();

                await js.deleteRecord("deleteStore", entity.Id);
            }
        }

        hasError = false;
        pending = 0;
    }

    catch(Exception ex) {
        hasError = true;
    }
}
```

```
synchronizing = false;
StateHasChanged();
}
```

Agregamos la notificación luego de grabar un registro en nuestra IndexedDB:

CreateGenre.razor

```
@page "/genres/create"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
@inject IJSRuntime js
@inject PendingState pendingState

<h3>Create Genre</h3>

<GenreForm @ref="genreForm" Genre="genre" OnValidSubmit="Create" />

@code {
    private Genre genre = new Genre(); //No se puede null, creamos un objeto vacío
    private GenreForm? genreForm;

    //No podemos llamarlo CreateGenre porque habría colisión con el nombre del componente.
    private async Task Create() {
        try {
            var httpResponse = await repository.Post("api/genres", genre);

            if (httpResponse.Error) {
                var ErrMessage = await httpResponse.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
            else {
                genreForm!.formPostedCorrectly = true;
                navManager.NavigateTo("/genres");
            }
        }
        catch(Exception ex) {
            if(ex.GetType().ToString() == "System.Net.Http.HttpRequestException") {
                await js.saveCreateRecord("api/genres", genre);
                await pendingState.NotifyUpdatePendingSynchronizations();
                genreForm!.formPostedCorrectly = true;
                navManager.NavigateTo("/genres");
            }
            else
        }
    }
}
```

```
        throw ex;  
    }  
}
```

Borrando géneros offline

A

GenresList.razor

```
@page "/genres"
@using Microsoft.AspNetCore.Authorization;
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
@inject IJSRuntime js
@inject PendingState pendingState

<h3>Genres</h3>

<div class="mb-3">
    <a class="btn btn-info" href="/genres/create">Add genre</a>
</div>

<GenericList List="Genres">
    <HasRecordsComplete>
        <table class="table table-striped">
            <thead>
                <tr>
                    <th></th>
                    <th>Name</th>
                </tr>
            </thead>
            <tbody>
                @foreach(var item in Genres!) {
                    <tr>
                        <td>
                            <a href="/genres/edit/@item.ID" class="btn btn-success">Edit</a>
                            <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                        </td>
                        <td>@item.Name</td>
                    </tr>
                }
            </tbody>
        </table>
    </HasRecordsComplete>
</GenericList>

@code {
    public List<Genre>? Genres { get; set; }
```

```

protected async override Task OnInitializedAsync() {
    await Load();
}

private async Task Load() {
    var responseHTTP = await repository.Get<List<Genre>>("api/genres");
    Genres = responseHTTP.Response!;
}

public async Task Delete(Genre genre) {
    try {
        var responseHTTP = await repository.Delete($"api/genres/{genre.ID}");

        if(responseHTTP.Error) {
            if(responseHTTP.httpResponseMessage.StatusCode == System.Net.HttpStatusCode.NotFound)
                navManager.NavigateTo("/");
            else {
                var ErrMessage = await responseHTTP.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
        }
        else {
            await Load();
        }
    }
    catch(Exception ex) {
        if(ex.GetType().ToString() == "System.Net.Http.HttpRequestException") {
            await js.saveDeleteRecord("api/genres/{genre.ID}");
            await pendingState.NotifyUpdatePendingSynchronizations();
            Genres!.Remove(genre);
        }
        else
            throw ex;
    }
}
}

```

Hacemos lo mismo con actores:

CreateActor.razor

```

@page "/actors/create"
@using Microsoft.AspNetCore.Authorization;
@inject NavigationManager navManager
@inject IRepository repository
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
@inject IJSRuntime js
@inject PendingState pendingState

```

```
<h3>Create Actor</h3>

<ActorsForm OnValidSubmit="Create" Actor="Actor" />
@code {
    private Actor Actor = new Actor();

    async Task Create() {
        try {
            var httpResponse = await repository.Post("api/actors", Actor);

            if (httpResponse.Error) {
                var ErrMessage = await httpResponse.GetErrMsg();
                await swAl.FireAsync("Error", ErrMessage, SweetAlertIcon.Error);
            }
            else {
                navManager.NavigateTo("/actors");
            }
        }
        catch(Exception ex) {
            if(ex.GetType().ToString() == "System.Net.Http.HttpRequestException") {
                await js.saveCreateRecord("api/actors", Actor);
                await pendingState.NotifyUpdatePendingSynchronizations();
                navManager.NavigateTo("/actors");
            }
            else
                throw ex;
        }
    }
}
```

Logout.razor

```
@page "/actors"
@using Microsoft.AspNetCore.Authorization;
@inject IRepository repository
@inject NavigationManager navManager
@inject SweetAlertService swAl
@attribute [Authorize(Roles = "admin")]
@inject IJSRuntime js
@inject PendingState pendingState

<h3>Actors list</h3>

<div class="mb-3">
    <a href="actors/create" class="btn btn-info">Create actor</a>
</div>

<Pagination ActualPage="ActualPage"
```

```
TotalPages="TotalPages"
SelectedPage="SelectedPage" />

<GenericList List="Actors">
<HasRecordsComplete>
    <table class="table table-striped">
        <thead>
            <tr>
                <th></th>
                <th>Name</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var item in Actors!) {
                <tr>
                    <td>
                        <a href="actors/edit/@item.ID" class="btn btn-success">Edit</a>
                        <button class="btn btn-danger" @onclick="@(() => Delete(item))">Delete</button>
                    </td>
                    <td>&ampnbsp@item.Name</td>
                </tr>
            }
        </tbody>
    </table>
</HasRecordsComplete>
</GenericList>
@code {
    public List<Actor>? Actors { get; set; }
    private int ActualPage = 1;
    private int TotalPages;

    protected async override Task OnInitializedAsync() {
        await Load();
    }

    private async Task SelectedPage(int page) {
        ActualPage = page;
        await Load(page);
    }

    private async Task Load(int Page = 1) {
        var responseHTTP = await repository.Get<List<Actor>>($"api/actors?page={Page}");
        Actors = responseHTTP.Response!;
        TotalPages =
int.Parse(responseHTTP.httpResponseMessage.Headers.GetValues("totalPages").FirstOrDefault());
    }

    public async Task Delete(Actor actor) {
        try {
            var responseHTTP = await repository.Delete($"api/actors/{actor.ID}");
        }
    }
}
```

```
if (responseHTTP.Error) {
    if (responseHTTP.httpResponseMessage.StatusCode ==
System.Net.HttpStatusCode.NotFound)
        navManager.NavigateTo("/");
    else {
        var ErrMessage = await responseHTTP.GetErrMsg();
        await swal.fire("Error", ErrMessage, SweetAlertIcon.Error);
    }
}
else {
    await Load();
}
}

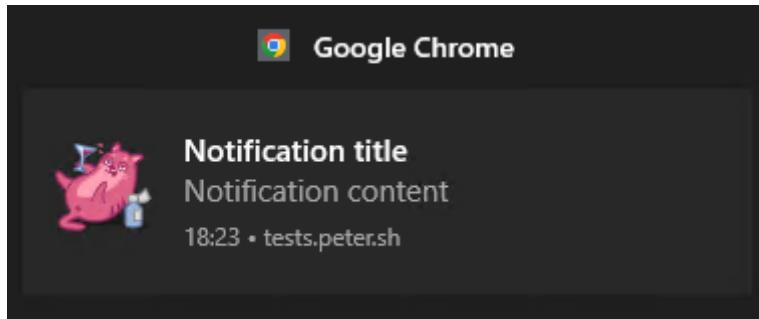
catch(Exception ex) {
    if(ex.GetType().ToString() == "System.Net.Http.HttpRequestException") {
        await js.saveDeleteRecord("api/actors/{actor.ID}");
        await pendingState.NotifyUpdatePendingSynchronizations();
        Actors!.Remove(actor);
    }
    else
        throw ex;
}
}
```

Notificaciones

Se pueden enviar notificaciones a nivel del navegador. De hecho, podemos recibirla mientras que nuestra app no esté corriendo.

Vamos a la página <https://tests.peter.sh/notification-generator/> para probar esto. Permitiremos las notificaciones.

Las partes básicas son: **Title**, **Body**, **Direction**, **Image**, **Icon** y **Badge**. Hacemos click en **Display notification** y vemos:



Push API – Back-End

Vimos que podemos ejecutar notificaciones a nivel del sistema operativo desde nuestras aplicaciones web. Ahora vamos a hacer es que vamos a utilizar el **Push API**. Con esto vamos a poder ponernos en contacto con los servidores de Google Chrome o Firefox para enviar un mensaje a sus servidores y éstos se encarguen de hacer que una notificación salga en el computador del usuario. Cuando nosotros utilizamos el **Push API** nosotros no somos los que enviamos la notificación directamente al dispositivo del usuario, sino que los servidores de Google Chrome, Firefox o cualquier navegador que use el usuario son los que se encargan de realizar la comunicación directa con el navegador para que así salga la notificación en el dispositivo del usuario, sea ese dispositivo una computadora o un celular, entonces colocaremos un botón que le va a permitir a los usuarios suscribirse al hecho de que haya nuevas películas en cartelera.

Creamos una clase **Notification.cs** en la carpeta **Entities** del proyecto **Shared**. Tanto P256dh como Auth serán informados por el navegador del usuario.

Notifcation.razor

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Notification {
        public int Id { get; set; }
        public string URL { get; set; }
        public string P256h { get; set; }
        public string Auth { get; set; }
    }
}
```

Agregamos la entidad a nuestro ApplicationDbContext.cs.

ApplicationDbContext.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : IdentityDbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder); //No se puede eliminar esta línea.
        }
    }
}
```

```

modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });
modelBuilder.Entity<MovieActor>().HasKey(x => new { x.ActorID, x.MovieID });
}

public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase
Genre.
public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase
Movie.
public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla
GenresMovie a partir de la clase GenresMovie.
public DbSet<MovieActor> MoviesActors => Set<MovieActor>(); //Creamos la tabla MoviesActors
a partir de la clase MovieActor.
public DbSet<VoteMovie> VotesMovies => Set<VoteMovie>(); //Creamos la tabla VotesMovies a
partir de la clase VoteMovie.
public DbSet<Notification> Notifications => Set<Notification>(); //Creamos la tabla Notifications a
partir de la clase Notification.
}
}

```

Creamos la migración y actualizamos:

PMC	EFC cli
Add-Migration Notification	dotnet ef migrations add Notification
Update-database	dotnet ef database update

```

Each
package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any
licenses to, third-party packages. Some packages may include dependencies which are governed by
additional licenses. Follow the package source (feed) URL to determine any dependencies.

```

```
Package Manager Console Host Version 6.5.0.154
```

```
Type 'get-help NuGet' to see all available NuGet commands.
```

```

PM> Add-Migration Notification
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> Update-Database
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (18ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT OBJECT_ID(N'[_EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    SELECT OBJECT_ID(N'[_EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]

```

```

Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
SELECT [MigrationId], [ProductVersion]
FROM [__EFMigrationsHistory]
ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20230224213801_Notification'.
Applying migration '20230224213801_Notification'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (24ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE TABLE [Notifications] (
        [Id] int NOT NULL IDENTITY,
        [URL] nvarchar(max) NOT NULL,
        [P256h] nvarchar(max) NOT NULL,
        [Auth] nvarchar(max) NOT NULL,
        CONSTRAINT [PK_Notifications] PRIMARY KEY ([Id])
    );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (6ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
    VALUES ('N'20230224213801_Notification', N'7.0.2');
Done.
PM>

```

Algo más que necesitamos son un par de llaves que nos van a servir para poder establecer el canal de comunicación con los servidores del navegador del usuario. Para eso podemos utilizar un servicio público:

<https://tools.reactpwa.com/vapid>

Escribimos el email y generamos la llave pública y la privada. Copiamos el objeto y lo pegamos en appSettings.json:

appSettings.json

```
{
    "ConnectionStrings": {
        "DefaultConnection": "Server=localhost;Database=BlazorPeliculas;TrustServerCertificate=True;Trusted_Connection=False;UserId=blazorpeliculasuser;Password=Password1!",
        "AzureStorage": "blablabla"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Information",
            "Microsoft.AspNetCore": "Warning"
        }
    },
    "AllowedHosts": "*",
    "jwtkey": "utLEZPsC96sxppPqo5KtvI8Gm1VpoiGdVfdOZvcdpVgBcMbNLAxI7M9bsRwW1PUBTY05vBh5laA5bXw9MXyJNUg3SOPDZWpCEh1587RZ2OOomLGTK0kPKLJtFxR7k1",
    "notifications": {
        "subject": "mailto:yo@aca.com",
        "publicKey": "BIAQ3AUkjshLEOz7W7PrvtLOnHbXnUkhhDSC1Qfx4kOVknyH_s4ZUyGKaLOi0KMvKj70TwJ_LewdKOOhPknEasc"
    }
}
```

```
    "privateKey": "3Iz1j-Y272Mx65qxcUe9sTrO4DoEdIfByw9g1OZ0olw"
}
```

Crearemos una clase **NotificationsService.cs** en la carpeta **Helpers** del proyecto **Server** para crear un servicio a través del cual nos comunicaremos con los servidores de los navegadores.

Instalamos le paquete **NuGet** llamado **WebPush** de Cory Thompson en el proyecto **Server**.

NotificationsService.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.EntityFrameworkCore;
using System.Text.Json;
using WebPush;

namespace BlazorPeliculas.Server.Helpers {
    public class NotificationsService {
        private readonly IConfiguration configuration;
        private readonly ApplicationDbContext context;

        public NotificationsService(IConfiguration configuration, ApplicationDbContext context) {
            this.configuration = configuration;
            this.context = context;
        }

        public async Task SendNotificationMovieOnBoard(Movie movie) {
            var notifications = await context.Notifications.ToListAsync();

            var publicKey = configuration.GetValue<string>("notifications:publicKey");
            var privateKey = configuration.GetValue<string>("notifications:privateKey");
            var subject = configuration.GetValue<string>("notifications:subject");

            var vapidDetails = new VapidDetails(subject, publicKey, privateKey);

            foreach(var notification in notifications) {
                var pushSubscription = new PushSubscription(notification.URL, notification.P256h,
                    notification.Auth);

                var webPushClient = new WebPushClient();

                try {
                    var payload = JsonSerializer.Serialize(new {
                        title = movie.Title,
                        image = movie.Poster,
                        url = $"movie/{movie.ID}/{movie.urlTitle()}"
                    });
                
```

```
        await webPushClient.SendNotificationAsync(pushSubscription, payload, vapidDetails);
    }
    catch(Exception ex) {
        throw ex;
    }
}
}
}
```

Registramos el servicio en **Program.cs** del proyecto **Server**. Debemos permitirle, también, al front-end recuperar la info de la llave pública.

Program.cs

```
using BlazorPeliculas.Server;
using BlazorPeliculas.Server.Helpers;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.ResponseCompression;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using System.Text.Json.Serialization;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews()
    .AddJsonOptions(options => options.JsonSerializerOptions.ReferenceHandler =
ReferenceHandler.IgnoreCycles);
builder.Services.AddRazorPages();

builder.Services.AddDbContext<ApplicationDbContext>(options =>
options.UseSqlServer("name=DefaultConnection"));
builder.Services.AddTransient<IFileSaver, FileSaverLocal>();
builder.Services.AddHttpContextAccessor();
builder.Services.AddAutoMapper(typeof(Program));
builder.Services.AddIdentity<IdentityUser, IdentityRole>()
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddDefaultTokenProviders();

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
        options.TokenValidationParameters = new TokenValidationParameters {
            ValidateIssuer = false,
            ValidateAudience = false,
            ValidateLifetime = true,
```

```
ValidateIssuerSigningKey = true,
IssuerSigningKey = new SymmetricSecurityKey(
    Encoding.UTF8.GetBytes(builder.Configuration["jwtkey"]!)),
ClockSkew = TimeSpan.Zero
}
);
builder.Services.AddScoped<NotificationsService>();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseWebAssemblyDebugging();
}
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.UseBlazorFrameworkFiles();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();
app.UseAuthorization();

app.MapRazorPages();
app.MapControllers();
app.MapGet("/api/config/notificationspublickey", async context => {
    var configuration = context.RequestServices.GetRequiredService< IConfiguration>();
    var publicKey = configuration.GetValue<string>("notifications:publicKey");
    await context.Response.WriteAsync(publicKey!);
});
app.MapFallbackToFile("index.html");

app.Run();
```

Crearemos un controlador que agregue y borre registros de nuestra table de notificaciones. Creamos la clase **NotificationsController.cs** en la carpeta **Controllers** del proyecto **Server**. La hacemos heredar de **ControllerBase**.

NotificationsController.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/notifications"), ApiController]
    public class NotificationsController: ControllerBase {
        private readonly ApplicationDbContext context;

        public NotificationsController(ApplicationDbContext context) {
            this.context = context;
        }

        [HttpPost("suscribe")]
        public async Task<ActionResult> Suscribe(Notification notification) {
            context.Add(notification);
            await context.SaveChangesAsync();
            return NoContent();
        }

        [HttpPost("unsuscribe")]
        public async Task<ActionResult> Unsuscribe(Notification notification) {
            var notificationDB = context.Notifications
                .FirstOrDefault(x => x.Auth == notification.Auth && x.P256h == notification.P256h);

            if (notificationDB == null) return NotFound();

            context.Remove(notificationDB);
            await context.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

Push API I – Front-End

Agregamos un método de js para saber si tenemos permiso para mandar notificaciones:

Utilities.js

```
function DotNetStaticTest() {
    DotNet.invokeMethodAsync("BlazorPeliculas.Client", "GetCurrentCount")
        .then(result => {
            console.log('Count from js: ' + result);
        });
}

function DotNetInstanceTest(dotNetHelper) {
    dotNetHelper.invokeMethodAsync("IncrementCount");
}
```

```
function inactiveTimer(dotnetHelper) {
    var timer;

    document.onmousemove = resetimer;
    document.onkeypress = resetimer;

    function resetimer() {
        clearTimeout(timer);
        timer = setTimeout(logout, 5 * 60 * 1000); // 5 minutos
    }

    function logout() {
        dotnetHelper.invokeMethodAsync("Logout");
    }
}

var db = new Dexie("mydb");
var dbVersion = 1;

db.version(dbVersion).stores({
    createStore: 'id++',
    deleteStore: 'id++'
});

async function getPendingRecords() {
    return await {
        ObjectsToCreate: await db.createStore.toArray(),
        ObjectsToDelete: await db.deleteStore.toArray()
    };
}

async function deleteRecord(table, id) {
    await db[table].where({ "id": id }).delete();
}

async function getPendingRecordCount() {
    var count = await db.createStore.count();
    count += await db.deleteStore.count();

    return count;
}

async function saveCreateRecord(url, body) {
    await db.createStore.put({ url, body: JSON.parse(body) });
}

async function saveDeleteRecord(url) {
    await db.deleteStore.put({ url });
}
```

```
async function getStatusNotificationGrant() {
    const grant = Notification.permission;

    if(grant === 'denied') return grant;

    //Estamos suscriptos?
    const worker = await navigator.serviceWorker.getRegistration();
    const existingSuscription = await worker.pushManager.getSubscription();

    const existingSuscription = await worker.pushManager.getSubscription();
    if (existingSuscription)
        return "granted";
    else
        return "default"; //Aún no contestó
}

async function suscribeUser() {
    var notificationGrant = await Notification.requestPermission();
    if (notificationGrant != 'granted') return null;

    const worker = await navigator.serviceWorker.getRegistration();
    const existingSuscription = await worker.pushManager.getSubscription();

    if (!existingSuscription) {
        const publicKeyResponse = await fetch('/api/config/noticationspublickey');
        const publicKey = await publicKeyResponse.text();

        const newSubscription = await worker.pushManager.subscribe({
            userVisibleOnly: true,
            applicationServerKey: publicKey
        });

        return buildSuscriptionResponse(newSubscription);
    }
    else {
        return buildSuscriptionResponse(existingSuscription);
    }
}

async function unsubscribeUser() {
    const worker = await navigator.serviceWorker.getRegistration();
    const existingSuscription = await worker.pushManager.getSubscription();

    if(existingSuscription) {
        existingSuscription.unsubscribe();
        return buildSuscriptionResponse(existingSuscription);
    }
}
```

```
function buildSuscriptionResponse(suscription) {
    return {
        url: suscription.endpoint,
        p256dh: arrayBufferToBase64(suscription.getKey('p256dh')),
        auth: arrayBufferToBase64(suscription.getKey('auth'))
    }
}

function arrayBufferToBase64(buffer) {
    // https://stackoverflow.com/a/9458996
    var binary = '';
    var bytes = new Uint8Array(buffer);
    var len = bytes.byteLength;
    for (var i = 0; i < len; i++) {
        binary += String.fromCharCode(bytes[i]);
    }
    return window.btoa(binary);
}
```

IJSRuntimeExtensionsMethods.js

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.JSInterop;
using System.Runtime.CompilerServices;
using System.Text.Json;

namespace BlazorPeliculas.Client.Helpers {
    public static class IJSRuntimeExtensionMethods {

        public static async ValueTask<string> GetStatusNotificationGrant(this IJSRuntime js) {
            return await js.InvokeAsync<string>("getStatusNotificationGrant");
        }

        public static async ValueTask<Notification> SuscribeUser(this IJSRuntime js) {
            return await js.InvokeAsync<Notification>("suscribeUser");
        }

        public static async ValueTask<Notification> UnsubscribeUser(this IJSRuntime js) {
            return await js.InvokeAsync<Notification>("unsubscribeUser");
        }

        public static async ValueTask<DbLocalRecords> GetPendingRecords(this IJSRuntime js) {
            return await js.InvokeAsync<DbLocalRecords>("getPendingRecords");
        }

        public static async ValueTask deleteRecord(this IJSRuntime js, string table, int id) {
            await js.InvokeVoidAsync("deleteRecord", table, id);
        }

        public static async ValueTask saveCreateRecord<T>(this IJSRuntime js, string url, T entity) {
```

```
var body = JsonSerializer.Serialize(entity);
await js.InvokeVoidAsync("saveCreateRecord", url, body);
}

public static async ValueTask saveDeleteRecord(this IJSRuntime js, string url) {
    await js.InvokeVoidAsync("saveDeleteRecord", url);
}

public static async ValueTask<int> GetPendingRecordCount(this IJSRuntime js) {
    return await js.InvokeAsync<int>("getPendingRecordCount");
}

public static async ValueTask<bool> Confirm(this IJSRuntime js, string message) {
    await js.InvokeVoidAsync("console.log", $"Asking: «{message}»");
    return await js.InvokeAsync<bool>("confirm", message);
}

public static ValueTask<object> SetInLocalStorage(this IJSRuntime js,
    string key, string value) {
    return js.InvokeAsync<object>("localStorage.setItem", key, value);
}

public static ValueTask<object> GetFromLocalStorage(this IJSRuntime js,
    string key) {
    return js.InvokeAsync<object>("localStorage.getItem", key);
}

public static ValueTask<object> RemoveFromLocalStorage(this IJSRuntime js,
    string key) {
    return js.InvokeAsync<object>("localStorage.removeItem", key);
}
}
```

Index.razor

```
@page "/"
@inject IRepository repository
@inject IJSRuntime js
@inject SweetAlertService swal

<PageTitle>Blazor Movies</PageTitle>

@if(notificationsGrant == "granted") {
    <button @onclick="Unsubscribe" class="btn btn-warning">Unsubscribe</button>
}
else if(notificationsGrant == "default") {
    <button @onclick="Subscribe" class="btn btn-primary">Subscribe</button>
}
<div>
    <h3>On billboard</h3>
    <div>
```

```

<MoviesList Movies="OnBoard">
    <Loading>
        
    </Loading>
    <NoRecords>
        <p>No movies to show</p>
    </NoRecords>
</MoviesList>
</div>
</div>
<div>
    <h3>Next releases</h3>
    <div>
        <MoviesList Movies="NextReleases">
            <Loading>
                
            </Loading>
            <NoRecords>
                <p>No movies to show</p>
            </NoRecords>
        </MoviesList>
    </div>
</div>

@code {
    public List<Movie>? OnBoard { get; set; }
    public List<Movie>? NextReleases { get; set; }
    private string notificationsGrant = string.Empty;

    protected override async Task OnInitializedAsync() {
        var responseHTTP = await repository.Get<HomePageDTO>("api/movies");
        if(responseHTTP httpResponseMessage.IsSuccessStatusCode) {
            OnBoard = responseHTTP.Response!.OnBoard;
            NextReleases = responseHTTP.Response!.NextReleases;
            notificationsGrant = await js.GetStatusNotificationGrant();
        }
        else
            Console.WriteLine(responseHTTP httpResponseMessage.StatusCode);
    }

    private async Task Suscribe() {
        var notification = await js.SuscribeUser();

        if(notification != null) {
            await repository.Post("api/notifications/suscribe", notification);
            notificationsGrant = await js.GetStatusNotificationGrant();
        }
    }
}

```

```

        await swal.fireAsync("Success", "You'll get a notification when a new movie is on billboard",
SweetAlertIcon.Success);
        StateHasChanged();
    }

private async Task Unsubscribe() {
    var notification = await js.UnsubscribeUser();

    if(notification != null) {
        await repository.Post("api/notifications/unsubscribe", notification);
        notificationsGrant = await js.GetStatusNotificationGrant();
        await swal.fireAsync("Success ", "You'll not get a notification when a new movie is on billboard
anymore", SweetAlertIcon.Success);
        StateHasChanged();
    }
}
    
```

Antes de empezar la prueba vemos que no hay notificaciones:

```

SELECT *
FROM Notifications
    
```

	Results	Messages		
	Id	URL	P256h	Auth
.0 %				

El botón de Suscribe no invocaba nunca al `HttpPost`. Después de dar muchas vueltas, descubrí que la entidad `Notification` tenía el campo `P256h` en vez de `P256dh`. Eso provocaba que el campo `P256dh` de la notificación creada estuviera siempre en `null`.

Decidí crear una nueva entidad `Notif.cs` y realizar los cambios

```

Notif.cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace BlazorPeliculas.Shared.Entities {
    public class Notif {
        public int Id { get; set; }
    }
}
    
```

```

public string URL { get; set; }
public string P256dh { get; set; }
public string Auth { get; set; }
}
}

```

ApplicationDbContext.cs

```

using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace BlazorPeliculas.Server {
    public class ApplicationDbContext : IdentityDbContext {
        public ApplicationDbContext(DbContextOptions options) : base(options) {

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder) {
            base.OnModelCreating(modelBuilder); //No se puede eliminar esta línea.

            modelBuilder.Entity<GenresMovie>().HasKey(x => new { x.GenreID, x.MovieID });
            modelBuilder.Entity<MovieActor>().HasKey(x => new { x.ActorID, x.MovieID });
        }

        public DbSet<Genre> Genres => Set<Genre>(); //Creamos la tabla Genres a partir de la clase
        Genre.
        public DbSet<Actor> Actors => Set<Actor>(); //Creamos la tabla Actors a partir de la clase Actor.
        public DbSet<Movie> Movies => Set<Movie>(); //Creamos la tabla Movies a partir de la clase
        Movie.
        public DbSet<GenresMovie> GenresMovie => Set<GenresMovie>(); //Creamos la tabla
        GenresMovie a partir de la clase GenresMovie.
        public DbSet<MovieActor> MoviesActors => Set<MovieActor>(); //Creamos la tabla MoviesActors
        a partir de la clase MovieActor.
        public DbSet<VoteMovie> VotesMovies => Set<VoteMovie>(); //Creamos la tabla VotesMovies a
        partir de la clase VoteMovie.
        public DbSet<Notification> Notifications => Set<Notification>(); //Creamos la tabla Notifications a
        partir de la clase Notification.
        public DbSet<Notif> Notifs => Set<Notif>(); //Creamos la tabla Notifs a partir de la clase Notif.
    }
}

```

PMC	EFC cli
Add-Migration Notifs	dotnet ef migrations add Notifs
Update-database	dotnet ef database update

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages. Some packages may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to determine any dependencies.

```
Package Manager Console Host Version 6.5.0.154

Type 'get-help NuGet' to see all available NuGet commands.

PM> Add-Migration Notifs
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> UPDATE-DATABASE
Build started...
Build succeeded.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (14ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (10ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT 1
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (0ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
        SELECT [MigrationId], [ProductVersion]
        FROM [__EFMigrationsHistory]
        ORDER BY [MigrationId];
Microsoft.EntityFrameworkCore.Migrations[20402]
    Applying migration '20230225053849_Notifs'.
    Applying migration '20230225053849_Notifs'.
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (9ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    CREATE TABLE [Notifs] (
        [Id] int NOT NULL IDENTITY,
        [URL] nvarchar(max) NOT NULL,
        [P256dh] nvarchar(max) NOT NULL,
        [Auth] nvarchar(max) NOT NULL,
        CONSTRAINT [PK_Notifs] PRIMARY KEY ([Id])
    );
Microsoft.EntityFrameworkCore.Database.Command[20101]
    Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
    INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
        VALUES (N'20230225053849_Notifs', N'7.0.3');
Done.
PM>
```

NotificationsService.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.EntityFrameworkCore;
using System.Text.Json;
using WebPush;

namespace BlazorPeliculas.Server.Helpers {
    public class NotificationsService {
        private readonly IConfiguration configuration;
        private readonly ApplicationDbContext context;

        public NotificationsService(IConfiguration configuration, ApplicationDbContext context)
        {
```

```
this.configuration = configuration;
this.context = context;
}

public async Task SendNotificationMovieOnBoard(Movie movie) {
    var notifications = await context.Notifs.ToListAsync();

    var publicKey = configuration.GetValue<string>("notifications:publicKey");
    var privateKey = configuration.GetValue<string>("notifications:privateKey");
    var subject = configuration.GetValue<string>("notifications:subject");

    var vapidDetails = new VapidDetails(subject, publicKey, privateKey);

    foreach(var notification in notifications) {
        var pushSubscription = new PushSubscription(notification.URL, notification.P256dh,
notification.Auth);

        var webPushClient = new WebPushClient();

        try {
            var payload = JsonSerializer.Serialize(new {
                title = movie.Title,
                image = movie.Poster,
                url = $"movie/{movie.ID}/{movie.urlTitle()}"
            });

            await webPushClient.SendNotificationAsync(pushSubscription, payload, vapidDetails);
        }
        catch(Exception ex) {
            throw ex;
        }
    }
}
}
```

NotificationsController.cs

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Mvc;

namespace BlazorPeliculas.Server.Controllers {
    [ApiController, Route("api/notifications")]
    public class NotificationsController: ControllerBase {
        private readonly ApplicationDbContext context;

        public NotificationsController(ApplicationDbContext context) {
            this.context = context;
        }
    }
}
```

```
[HttpPost("suscribe")]
public async Task<ActionResult> Suscribe(Notif notification) {
    context.Add(notification);
    await context.SaveChangesAsync();
    return NoContent();
}

[HttpPost("unsubscribe")]
public async Task<ActionResult> Unsubscribe(Notif notification) {
    var notificationDB = context.Notifs
        .FirstOrDefault(x => x.Auth == notification.Auth && x.P256dh == notification.P256dh);

    if (notificationDB == null) return NotFound();

    context.Remove(notificationDB);
    await context.SaveChangesAsync();
    return NoContent();
}
}
```

IJSRuntimeExtensionsMethods.js

```
using BlazorPeliculas.Shared.Entities;
using Microsoft.JSInterop;
using System.Runtime.CompilerServices;
using System.Text.Json;

namespace BlazorPeliculas.Client.Helpers {
    public static class IJSRuntimeExtensionMethods {

        public static async ValueTask<string> GetStatusNotificationGrant(this IJSRuntime js) {
            return await js.InvokeAsync<string>("getStatusNotificationGrant");
        }

        public static async ValueTask<Notif> SuscribeUser(this IJSRuntime js) {
            return await js.InvokeAsync<Notif>("suscribeUser");
        }

        public static async ValueTask<Notif> UnsubscribeUser(this IJSRuntime js) {
            return await js.InvokeAsync<Notif>("unsubscribeUser");
        }

        public static async ValueTask<DbLocalRecords> GetPendingRecords(this IJSRuntime js) {
            return await js.InvokeAsync<DbLocalRecords>("getPendingRecords");
        }

        public static async ValueTask deleteRecord(this IJSRuntime js, string table, int id) {
            await js.InvokeVoidAsync("deleteRecord", table, id);
        }
    }
}
```

```
public static async ValueTask saveCreateRecord<T>(this IJSRuntime js, string url, T entity) {
    var body = JsonSerializer.Serialize(entity);
    await js.InvokeVoidAsync("saveCreateRecord", url, body);
}

public static async ValueTask saveDeleteRecord(this IJSRuntime js, string url) {
    await js.InvokeVoidAsync("saveDeleteRecord", url);
}

public static async ValueTask<int> GetPendingRecordCount(this IJSRuntime js) {
    return await js.InvokeAsync<int>("getPendingRecordCount");
}

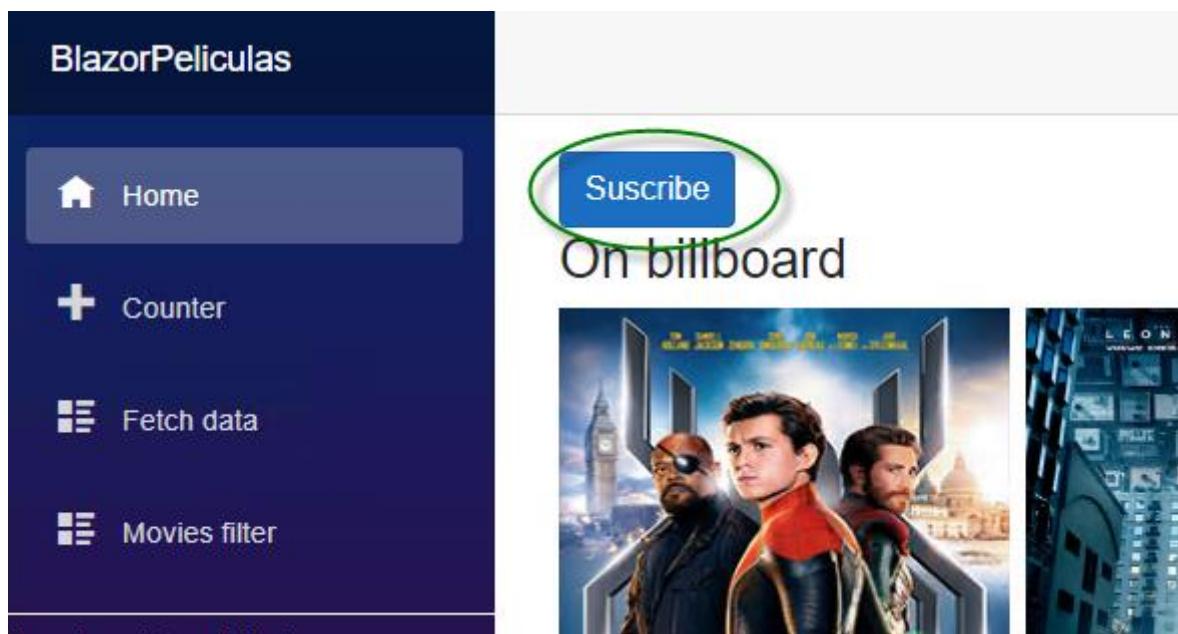
public static async ValueTask<bool> Confirm(this IJSRuntime js, string message) {
    await js.InvokeVoidAsync("console.log", $"Asking: «{message}»");
    return await js.InvokeAsync<bool>("confirm", message);
}

public static ValueTask<object> SetInLocalStorage(this IJSRuntime js,
    string key, string value) {
    return js.InvokeAsync<object>("localStorage.setItem", key, value);
}

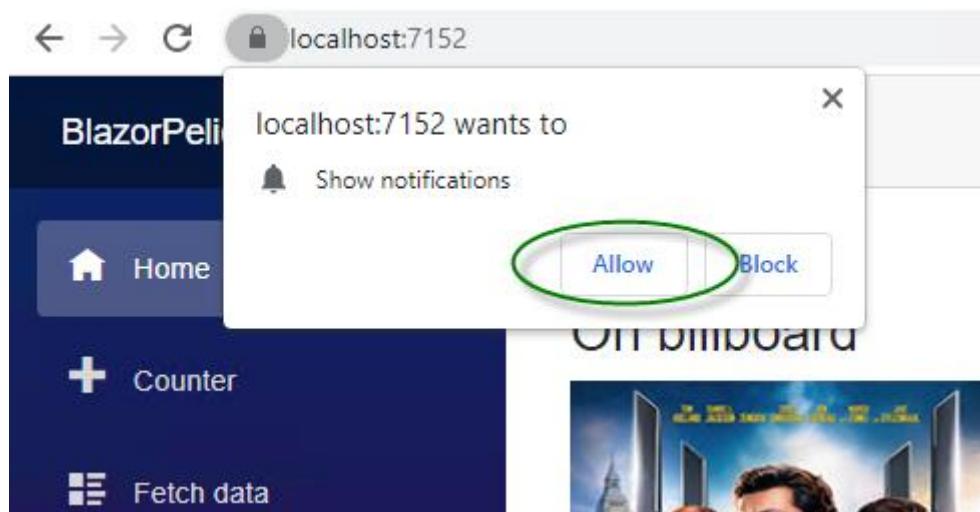
public static ValueTask<object> GetFromLocalStorage(this IJSRuntime js,
    string key) {
    return js.InvokeAsync<object>("localStorage.getItem", key);
}

public static ValueTask<object> RemoveFromLocalStorage(this IJSRuntime js,
    string key) {
    return js.InvokeAsync<object>("localStorage.removeItem", key);
}
```

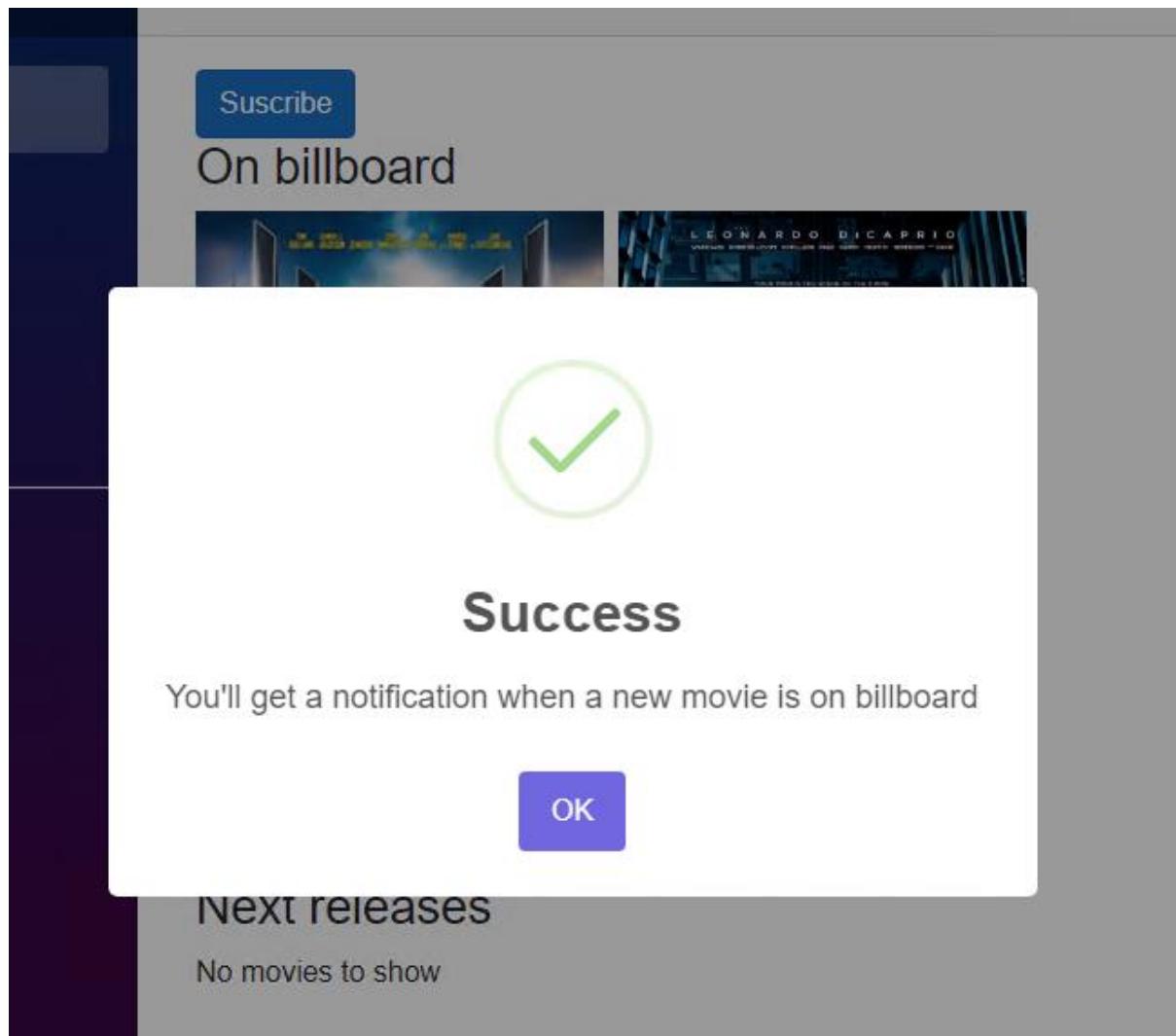
Ahora sí, las pruebas funcionan correctamente. Hacemos click en el botón de **Suscribe**:



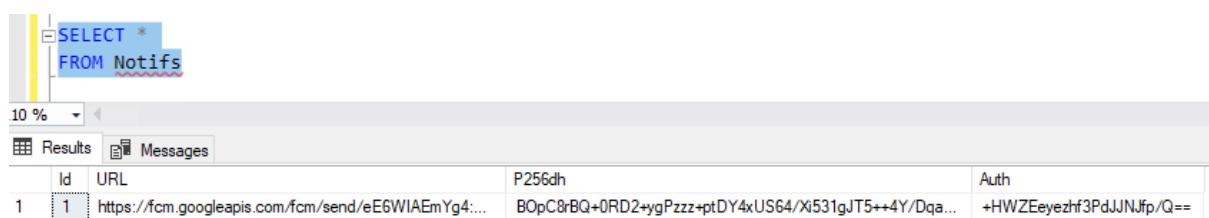
Y Chrome nos muestra el mensaje preguntando si queremos recibir notificaciones:



Se realiza la subscripción y recibimos el mensaje de éxito correspondiente:



En la tabla de Notifications ya disponemos de toda la info necesaria para poder enviar Push cuando sea necesario:



```
SELECT *
FROM Notifs
```

10 %

	Results	Messages		
	Id	URL	P256dh	Auth
1	1	https://fcm.googleapis.com/fcm/send/eE6WIAEmYg4...	B0pC8BQ+0RD2+ygPzz+ptDY4xUS64/Xi531gJT5++4Y/Dqa...	+HWZEeyezhf3PdJJNJfp/Q==

Ya estamos en condiciones de enviar mensajes. Para eso, tenemos que modificar la clase **MoviesController.cs** para que cuando haya una película nueva en cartelera se envíe la notificación a todo aquel que lo haya solicitado.

MoviesController.cs

```
using AutoMapper;
using BlazorPeliculas.Server.Helpers;
using BlazorPeliculas.Shared.DTOs;
using BlazorPeliculas.Shared.Entities;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.ComponentModel;

namespace BlazorPeliculas.Server.Controllers {
    [Route("api/movies"), ApiController]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme, Roles = "admin")]
    public class MoviesController : ControllerBase {
        private readonly ApplicationDbContext context;
        private readonly IFileSaver fileSaver;
        private readonly IMapper mapper;
        private readonly UserManager<IdentityUser> userManager;
        private readonly NotificationsService notificationsService;
        private readonly string container = "movies";

        public MoviesController(ApplicationDbContext context,
            IFileSaver fileSaver,
            IMapper mapper,
            UserManager<IdentityUser> userManager,
            NotificationsService notificationsService) {
            this.context = context;
            this.fileSaver = fileSaver;
            this.mapper = mapper;
            this.userManager = userManager;
            this.notificationsService = notificationsService;
        }

        [HttpGet, AllowAnonymous]
        public async Task<ActionResult<HomePageDTO>> Get() {
            var limit = 6;
            var onBoardMovies = await context.Movies
                .Where(movie => movie.OnBillboard)
                .Take(limit)
                .OrderByDescending(movie => movie.ReleaseDate)
                .ToListAsync();
            var today = DateTime.Today;
            var nextReleases = await context.Movies
                .Where(movie => movie.ReleaseDate > today)
                .Take(limit)
                .OrderBy(movie => movie.ReleaseDate)
                .ToListAsync();
        }
    }
}
```

```
var result = new HomePageDTO {
    OnBoard = onBoardMovies,
    NextReleases = nextReleases
};

return result;
}

[HttpGet("{id:int}"), AllowAnonymous]
public async Task<ActionResult<MovieViewDTO>> Get(int id) {
    var movie = await context.Movies
        .Where(movie => movie.ID == id)
        .Include(movie => movie.GenresMovie)
        .ThenInclude(gm => gm.Genre)
        .Include(movie => movie.MovieActor.OrderBy(ma => ma.Orden))
        .ThenInclude(ma => ma.Actor)
        .FirstOrDefaultAsync();

    if (movie is null) {
        //No se encontró la película
        return NotFound();
    }

    var votesMedia = 0.0;
    var userVote = 0;

    if(await context.VotesMovies.AnyAsync(x => x.MovieID == id)) {
        //Alguien ha votado por la película
        votesMedia = await context.VotesMovies
            .Where(x => x.MovieID == id)
            .AverageAsync(x => x.Voto);

        if(HttpContext.User.Identity!.IsAuthenticated) {
            var user = await userManager.FindByEmailAsync(HttpContext.User.Identity!.Name!);

            if(user is null) {
                //No debería pasar nunca, pero por las dudas...
                return BadRequest("User was not found");
            }

            var userID = user.Id;

            var userVoteDB = await context.VotesMovies
                .FirstOrDefaultAsync(x => x.MovieID == id && x.UserID == userID);

            if(userVoteDB is not null)
                userVote = userVoteDB.Voto;
        }
    }

    var model = new MovieViewDTO();
```

```
model.Movie = movie;
model.Genres = movie.GenresMovie.Select(gm => gm.Genre!).ToList();
model.Actors = movie.MovieActor.Select(ma => new Actor {
    Name = ma.Actor!.Name,
    Photo = ma.Actor.Photo,
    Character = ma.Character,
    ID = ma.Actor.ID
}).ToList();

model.VotesMedia = votesMedia;
model.UserVote = userVote;

return model;
}

[HttpGet("filter"), AllowAnonymous]
public async Task<ActionResult<List<Movie>>> Get([FromQuery] SearchMoviesParametersDTO
model) {
    var queryableMovies = context.Movies.AsQueryable();

    if(!string.IsNullOrWhiteSpace(model.Title))
        queryableMovies = queryableMovies
            .Where(x => x.Title.Contains(model.Title));

    if(model.Onbillboard)
        queryableMovies = queryableMovies
            .Where(x => x.OnBillboard);

    if(model.Releases) {
        var today = DateTime.Today;

        queryableMovies = queryableMovies
            .Where(x => x.ReleaseDate >= today);
    }

    if (model.GenreID != 0)
        queryableMovies = queryableMovies
            .Where(x => x.GenresMovie
                .Select(y => y.GenreID)
                .Contains(model.GenreID));

    if(model.MostVoted) {
        queryableMovies = queryableMovies.OrderByDescending(m => m.VotesMovies.Average(vm
=> vm.Voto));
    }

    await HttpContext.InserPaginationParametersInResponse(queryableMovies,
model.RecordCount);

    //Recién acá materealizo el query y lo ejecuto en la BD -> ejecución diferida
```

```
var movies = await queryableMovies.ToPage(model.pagination).ToListAsync();
return movies;
}

[HttpGet("edit/{id:int}")]
public async Task<ActionResult<MovieUpdateDTO>> PutGet(int id) {
    //Re-utilizamos el GET para traer el ActionResult con la info de la película.
    var movieActionResult = await Get(id);

    if (movieActionResult.Result is NotFoundResult)
        return NotFound();

    var movieViewDTO = movieActionResult.Value;      //Será el DTO
    var selectedGenresIDs = movieViewDTO!.Genres.Select(x => x.ID).ToList();
    var unselectedGenres = await context.Genres
        .Where(x => !selectedGenresIDs.Contains(x.ID))
        .ToListAsync();

    var model = new MovieUpdateDTO();
    model.Movie = movieViewDTO.Movie;
    model.UnselectedGenres = unselectedGenres;
    model.SelectedGenres = movieViewDTO.Genres;
    model.Actors = movieViewDTO.Actors;

    return model;
}

[HttpPost]
public async Task<ActionResult<int>> Post(Movie movie) {
    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        var poster = Convert.FromBase64String(movie.Poster);
        movie.Poster = await fileSaver.SaveFile(poster, "jpg", container);
    }

    WriteActorsOrder(movie);

    context.Add(movie);
    await context.SaveChangesAsync();
    return movie.ID;
}

private static void WriteActorsOrder(Movie movie) {
    if (movie.MovieActor is not null) {
        for (int i = 0; i < movie.MovieActor.Count; i++) {
            movie.MovieActor[i].Orden = i + 1;
        }
    }
}

[HttpPut]
```

```
public async Task<ActionResult> Put(Movie movie) {
    var movieDB = await context.Movies
        .Include(x => x.GenresMovie)
        .Include(x => x.MovieActor)
        .FirstOrDefaultAsync(x => x.ID == movie.ID);

    if (movieDB is null)
        return NotFound();

    var sendNotification = movie.OnBillboard == true || movieDB.OnBillboard == false;

    //Tomá las propiedades de movie y pasalas a movieDB
    movieDB = mapper.Map(movie, movieDB);

    if (!string.IsNullOrWhiteSpace(movie.Poster)) {
        //Nos mandaron una foto desde el frontend
        var Poster = Convert.FromBase64String(movie.Poster);
        movieDB.Poster = await fileSaver.EditFile(Poster, ".jpg", container, movieDB.Poster!);
    }

    WriteActorsOrder(movieDB);

    await context.SaveChangesAsync(); //Se hace el UPDATE

    if(sendNotification)
        await notificationsService.SendNotificationMovieOnBoard(movieDB);

    return NoContent(); //Todo se hizo correctamente
}

[HttpDelete("{id:int}")]
public async Task<ActionResult<int>> Delete(int id) {
    var movie = await context.Movies.FirstOrDefaultAsync(x => x.ID == id);

    if (movie is null)
        return NotFound();

    context.Remove(movie); //Marcamos para borrar el actor
    await context.SaveChangesAsync();
    if (!string.IsNullOrWhiteSpace(movie.Poster))
        await fileSaver.DeleteFile(movie.Poster!, container);

    return NoContent(); //Todo se hizo correctamente
}
```

Para poder recibir notificaciones Push, necesitaremos agregar código en el SW para recibir el mensaje correspondiente enviado por el navegador. Para eso, escuchamos el evento push.

También tenemos que escuchar el evento click para saber cuando el usuario clickea en una notificación.

service-worker.js

```
// Caution! Be sure you understand the caveats before publishing an application with
// offline support. See https://aka.ms/blazor-offline-considerations

self.importScripts('./service-worker-assets.js');
self.addEventListener('install', event => event.waitUntil(onInstall(event)));
self.addEventListener('activate', event => event.waitUntil(onActivate(event)));
self.addEventListener('fetch', event => event.respondWith(onFetch(event)));

const cacheNamePrefix = 'offline-cache-';
const cacheName = `${cacheNamePrefix}${self.assetsManifest.version}`;
const cacheNameDynamic = 'dynamic-cache';
const offlineAssetsInclude = [/\.dll$/, /\.pdb$/, /\.wasm/, /\.html/, /\.js$/, /\.json$/, /\.css$/, /\.woff$/,
/\.png$/, /\.jpe?g$/, /\.gif$/, /\.ico$/, /\.blat$/, /\.dat$/];
const offlineAssetsExclude = [/^service-worker\.js$/];

async function onInstall(event) {
    console.info('Service worker: Install');

    // Fetch and cache all matching items from the assets manifest
    const assetsRequests = self.assetsManifest.assets
        .filter(asset => offlineAssetsInclude.some(pattern => pattern.test(asset.url)))
        .filter(asset => offlineAssetsExclude.some(pattern => pattern.test(asset.url)))
        .map(asset => new Request(asset.url, { integrity: asset.hash, cache: 'no-cache' }));
    await caches.open(cacheName).then(cache => cache.addAll(assetsRequests));
}

async function onActivate(event) {
    console.info('Service worker: Activate');

    // Delete unused caches
    const cacheKeys = await caches.keys();
    await Promise.all(cacheKeys
        .filter(key => key.startsWith(cacheNamePrefix) && key !== cacheName)
        .map(key => caches.delete(key)));
}

async function onFetch(event) {
    if (event.request.method !== 'GET') {
        return fetch(event.request);
    }

    let cachedResponse = null;
```

```
// For all navigation requests, try to serve index.html from cache
// If you need some URLs to be server-rendered, edit the following check to exclude those URLs
const shouldServeIndexHtml = event.request.mode === 'navigate';

const request = shouldServeIndexHtml ? 'index.html' : event.request;
const cache = await caches.open(cacheName);
cachedResponse = await cache.match(request);

if(cachedResponse) {
    //Existe información estática cacheara. La retornaremos.
    return cachedResponse;
}

var response = await getAndUpdate(event);

return response;
}

async function getAndUpdate(event) {
    try {
        const response = await fetch(event.request);
        const contentType = response.headers.get('content-type');

        let saveInCache = true;
        if(contentType)
            saveInCache = !contentType.includes('text/html');

        if(saveInCache) {
            const cache = await caches.open(cacheNameDynamic);
            await cache.put(event.request, response.clone());
        }

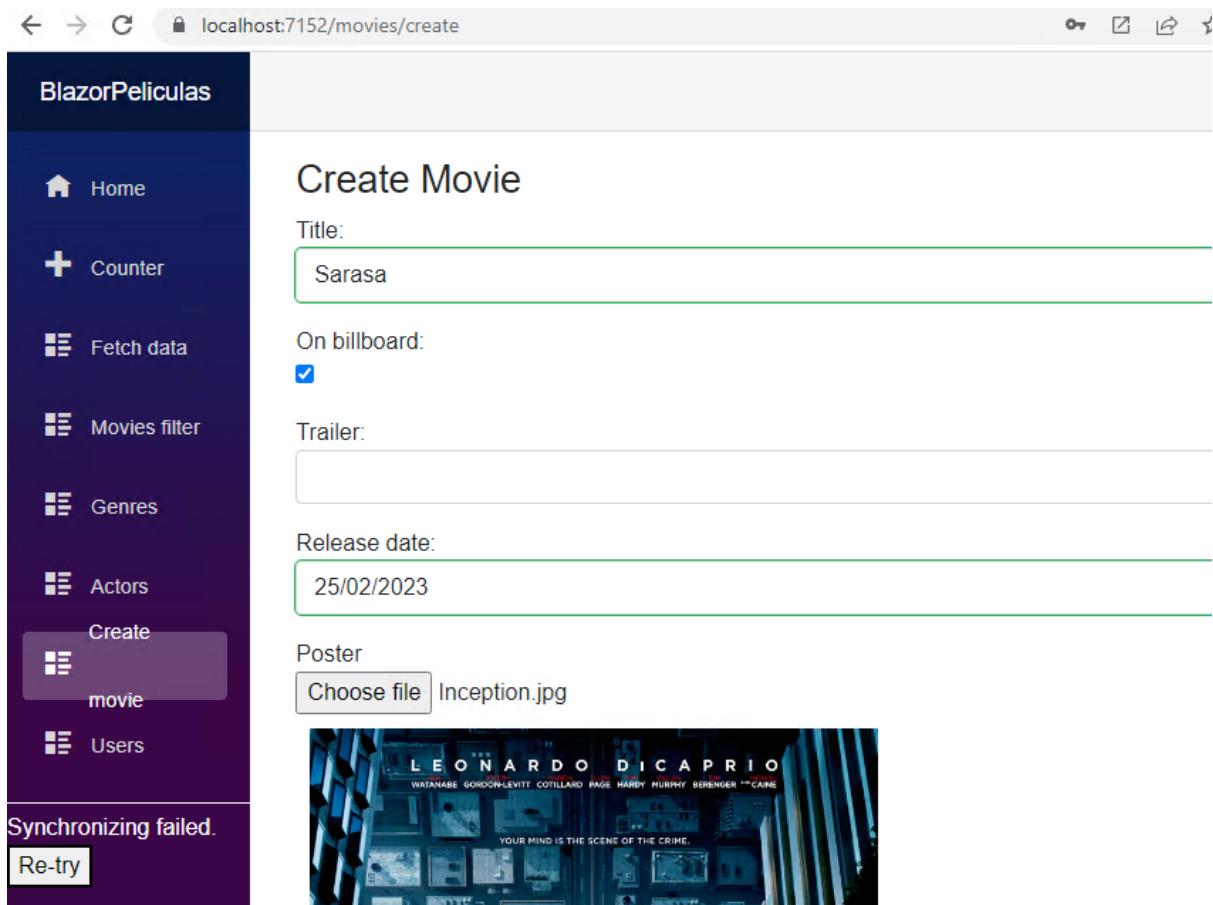
        return response;
    }
    catch {
        //Si hay un error, no pudimos establecer la conexión
        const cache = await caches.open(cacheNameDynamic);
        return cache.match(event.request);
    }
}

self.addEventListener('push', event => {
    const payload = event.data.json(); //info que le mandamos al navegador a través del payload.

    event.waitUntil(
        self.registration.showNotification('New movie on cinemas', {
            body: payload.title,
            image: payload.image,
            data: {url: payload.url}
        }));
})
```

```
});  
});  
  
self.addEventListener('notificationclick', event => {  
    event.notification.close();  
    event.waitUntil(clients.openWindow(event.notification.data.url));  
});
```

Crearemos una nueva película en cartelera:



localhost:7152/movies/create

BlazorPelículas

Create Movie

Title: Sarasa

On billboard:

Trailer:

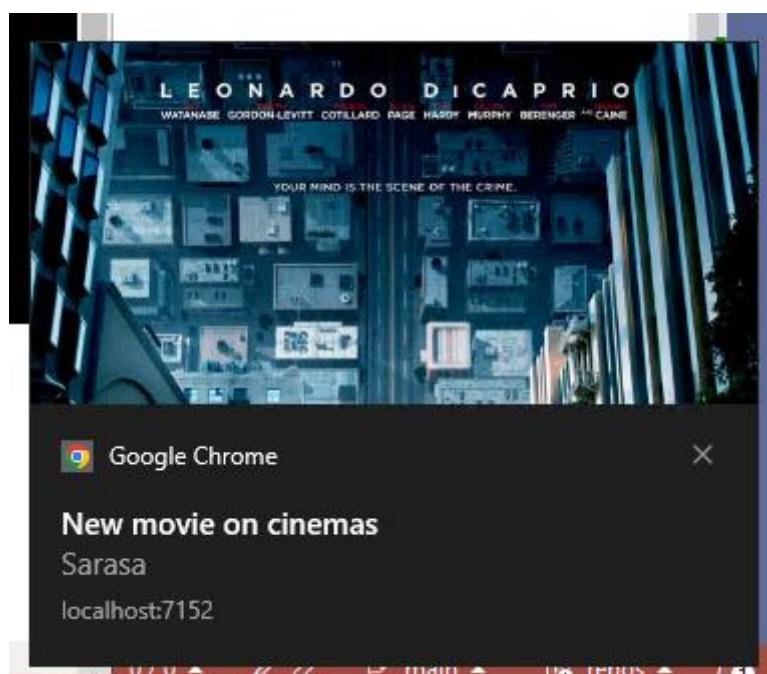
Release date: 25/02/2023

Poster

Choose file Inception.jpg

LEONARDO DICAPRIO
WATANABE GORDON-LEVITT COTILLARD PAGE HARDY MURPHY BERENGER COOGAN
YOUR MIND IS THE SCENE OF THE CRIME.

Al grabar, nos llega la notificación.



Para evitar problemas con los acentos, lo que hace es grabar el service-worker.js como Encoding UTF-8 (save as).

