Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Grado

Desarrollo de aplicación web para valoración y recomendación de películas, series y libros utilizando Spring Boot y ReactJS

Autor: Alberto González Martínez

Tutor/es: Salvador Otón Tortosa

2022

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

DESARROLLO DE APLICACIÓN WEB PARA VALORACIÓN Y RECOMENDACIÓN DE PELÍCULAS, SERIES Y LIBROS UTILIZANDO SPRING BOOT Y REACTJS

Alberto González Martínez

09 / 2022 Presentación

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Carrera

DESARROLLO DE APLICACIÓN WEB PARA VALORACIÓN Y RECOMENDACIÓN DE PELÍCULAS, SERIES Y LIBROS UTILIZANDO SPRING BOOT Y REACTJS

Autor: **Alberto González Martínez**

Director: **Salvador Otón Tortosa**

Tribunal:

Presidente: _____

Vocal 1º: _____

Vocal 2º: _____

Calificación: _____

Alcalá de Henares a de de 2022

A Lúa, por haber sido mi compañera durante muchos años.

A mis padres y mi hermana, por haberme guiado, enseñado y dado la mejor educación posible.

A mis amigos/as, por estar ahí cuando lo he necesitado.

A Marta por apoyarme y creer en mí.

Muchas gracias.

INDICE RESUMIDO

INDICE DETALLADO

INDICE DE FIGURAS

Resumen

Este proyecto tiene como finalidad el desarrollo de una aplicación web que permita a los usuarios consultar información, valorar, recomendar y marcar como favoritos libros, series, películas. Además, los usuarios podrán consultar también, la información de los actores y autores o directores.

El desarrollo de la aplicación es full-stack, es decir, se ha desarrollado tanto la parte gráfica e interfaz, o front-end, como la funcionalidad completa de esta, o back-end.

Para el desarrollo de la aplicación, se han utilizado tecnologías actuales, como son, por ejemplo, Spring Boot, para el API REST, MySQL, para la base de datos y ReactJS, para la aplicación web o interfaz de usuario. Por otro lado, también se han utilizado HTML5, CSS3, Bootstrap.

La aplicación final tiene un diseño responsivo o adaptable, lo que permite al usuario, acceder y navegar la aplicación desde cualquier dispositivo, ya sea un ordenador, un portátil, una tablet o un móvil.

Palabras clave

Spring Boot, ReactJS, HTML5, CSS3, Diseño Responsive, API REST, Full-stack

Abstract

The purpose of this project is to develop a web application that allows users to consult information, rate, recommend and mark books, series and movies as favorites. In addition, users will also be able to consult information about actors and authors or directors.

The development of the application is full-stack, that is, it has been developed both the graphical part and interface, or front-end, and the full functionality of it, or back-end.

For the development of the application, current technologies have been used, such as Spring Boot, for the REST API, MySQL, for the database and ReactJS, for the web application or user interface. On the other hand, HTML5, CSS3, Bootstrap have also been used.

The final application has a responsive or adaptive design, which allows the user to access and navigate the application from any device, whether a computer, a laptop, a tablet or a cell phone.

Keywords

Spring Boot, ReactJS, HTML5, CSS3, Responsive Design, REST API, Full-stack

1. Introducción

Bomose es una página web para que cualquier usuario pueda consultar datos o información, evaluar o compartir sus gustos y opiniones sobre películas, series o libros con otros usuarios que utilicen la página web.

Hoy en día, muchos usuarios, ya sea desde un ordenador, móvil o tablet, tienen acceso a internet y a un navegador. Esto ha sido uno de los motivos por los cuales se ha optado por realizar una página web, ya que es multiplataforma y no hay necesidad de crear una aplicación específica para los diferentes sistemas operativos o dispositivos.

Las series, películas y libros son uno de los medios de entretenimiento más consumidos actualmente, pero, en muchas ocasiones, es complicado elegir que ver o que leer, debido a la gran oferta que hay en internet y las nuevas producciones o títulos que se estrenan.

Por ello, es de gran utilidad, tener una página web como Bomose, que reúne la información, categorías y calificación de diferentes series, películas o libros en un mismo sitio.

El nombre Bomose es la unión de las primeras dos letras de las palabras en inglés, *books* (libros), *movies* (películas), *series* (series).

Para el desarrollo del proyecto, se ha tenido en cuenta en todo momento al usuario de la aplicación, haciendo que Bomose sea una página web amigable, fácil de entender y proporcione al usuario de todas las funcionalidades básicas que se esperarían de una página web de este tipo.

Además, en Bomose, no es necesario iniciar sesión o registrarse, con lo que, será accesible y atractiva para muchos más usuarios, los cuales únicamente quieran consultar información como:

- Datos sobre una película, serie o libro.
- Datos sobre actores o autores.
- Opiniones de usuarios.
- Nota media de los usuarios.

2. Objetivos

El objetivo principal del trabajo es el **desarrollo *full-stack* de una aplicación web** para la **evaluación y recomendación de películas, series y libros**, la cual, tenga una interfaz y usabilidad amigables para el usuario, empleando tecnologías altamente usadas a días de hoy, como lo son, en este caso, Spring Boot, ReactJS y MySQL.

En la parte ***back-end*** del proyecto se utilizarán Spring Boot, para crear un API REST y MySQL, para la base de datos.

En la parte ***front-end*** del proyecto se utilizarán HTML, CSS, ReactJS con diferentes librerías, Bootstrap, SASS.

Para conseguir lo anteriormente mencionado, se han de plantear y desarrollar los siguientes objetivos específicos:

1. **Estudiar las tecnologías que se quieren emplear**, para tener una visión global de las posibilidades de las que se disponen y conseguir un producto final robusto y que cumpla lo esperado.
2. **Evaluar las páginas web más utilizadas** que se encuentran dentro del mismo mercado, para identificar y evaluar aspectos que deberían y no deberían incluirse en la aplicación web.
3. **Crear un API REST con Spring Boot** que se comunique con la página web y permita realizar consultas de datos y operaciones CRUD.
4. **Crear un proyecto web en ReactJS** para mostrar los datos y hacer accesibles las operaciones del API REST mediante una interfaz.
5. **Crear una interfaz gráfica** que sea accesible para el usuario y que cuente con un diseño llamativo.
6. **Diseño responsive** para que la página web sea navegable en distintos dispositivos, independientemente de las dimensiones de este.
7. **Crear un manual de usuario** detallado sobre las funcionalidades y navegación de la aplicación web.
8. **Redacción de una memoria final**, en donde se explicará en detalle estructura, funcionamiento, diseño y desarrollo del proyecto.

3. Estado del arte

Una de las cosas más importantes a la hora de empezar desarrollo, es estudiar el mercado al que pertenece nuestro proyecto, para poder conseguir una visión general de los estándares a los que hay que ceñirse y las carencias que hay suplir.

Otro pilar fundamental a la hora de iniciar el desarrollo es conocer y estudiar las tecnologías que están presentes en el mercado, para elegir la que más se adapte a nuestras necesidades, facilite el desarrollo del proyecto y tenga soporte y compatibilidad con los distintos navegadores del mercado.

3.1. Páginas web de crítica de cine, series, libros

Hoy en día, debido a la gran oferta y fácil acceso que hay en el mercado de contenido en *streaming*, con numerosas plataformas y una gran cantidad de series, películas y libros, los usuarios necesitan saber que **contenido puede ajustarse a sus gustos y preferencias**, o que **contenido es recomendado por otros usuarios** de las plataformas o "merece la pena" ver.

A continuación, se muestran gráficas de Google Trends (https://trends.google.com/trends/) sobre la **popularidad de las plataformas de *streaming*** más conocidas y usadas, en las búsquedas en Google, desde el 2004 hasta día de hoy.

1. Popularidad del término "Netflix" en búsquedas de Google

2. Popularidad del término "HBO" en búsquedas de Google

3. Popularidad del término "Prime Video" en búsquedas de Google

Como se puede ver en las tres imágenes anteriores, la **popularidad** de las **plataformas *streaming***, y, por ende, del **consumo** del contenido de **series y películas** ha ido **aumentando considerablemente durante los años.**

Además, debido al gran catálogo que estas ofrecen, es complicado elegir o ver todas las series o películas que se ofrecen, por ello, las páginas web de valoración y crítica de series o películas, son de gran utilidad para muchos de los usuarios de estas plataformas a la hora de escoger que quieren ver.

Las páginas web de valoración y crítica de cine, series y/o libros, más usadas e importantes actualmente son las siguiente:

- **IMDb** (https://www.imdb.com/): es una de las páginas web más importantes a nivel mundial, que cuenta con la base de datos más grande del mundo sobre películas, series, programa de televisión, eventos en vivo…
- **Rotten Tomatoes** (https://www.rottentomatoes.com/): es una página web estadounidense de crítica y valoración de cine, series y televisión.

- **Sensacine** (https://www.sensacine.com/): es una página web española de crítica, valoración y actualidad sobre series y películas.
- **FilmAffinity** (https://www.filmaffinity.com/): es una página web española de valoración y crítica de series y películas.
- **Goodreads** (https://www.goodreads.com/): es una de las páginas de valoración de libros más importantes e influyentes del mundo.

Para el desarrollo de este proyecto se han tenido en cuenta, las distintas funcionalidades y diseño de estas páginas web para adaptar y utilizar los aspectos que se han considerado más importantes.

3.2. Tecnologías

En este apartado, se van a explicar las **principales tecnologías** que se han utilizado para **realizar el desarrollo del proyecto**. Desde las tecnologías *back-end* de la aplicación, como las que conforman la parte *front-end*.

3.2.1. Spring Boot

Una de las tecnologías que mayor peso tiene en el proyecto es Spring Boot, esta herramienta, permite crear aplicaciones Spring de una forma rápida y sencilla, evitando tener que configurar de forma manual la aplicación.

Se ha elegido el *framework* Spring por numerosas razones:

- **Gestiona de forma automática las operaciones CRUD** (Create, Read. Retriev, Update, Delete. Destroy) de la base de datos.
- Usa el **patrón de diseño MVC** (modelo vista controlador), el cual divide y separa la lógica de las diferentes clases y/o partes de la aplicación, lo que proporciona un mejor mantenimiento de esta y ayuda a la reutilización de los componentes.
- Realiza la **creación** y **mantenimiento** de la **base de datos de manera autónoma**, sin necesidad de tener que crear una base de datos.

- Los **datos de la aplicación** se **almacenan en objetos Java** lo cual, hace más fácil el acceso y manejo de estos.
- Es una **herramienta gratuita**.

Spring Boot ha sido usado para **desarrollar una API REST**, la cual se encarga de gestionar e implementar todas operaciones CRUD, consultas a la base de datos y diferentes funcionalidades, para, de una forma fácil y rápida, poder acceder a ellas desde la aplicación.

El acceso a las **funcionalidades del API REST**, desde la parte de la aplicación web, se realizan mediante diferentes **llamadas a los *endpoints* del API**. Esta comunicación se establece mediante diferentes peticiones HTTP, las peticiones que se han usado en este caso han sido las siguientes:

- **GET**: las peticiones de tipo GET, son utilizadas para **solicitar un recurso o recuperar datos.** Este tipo de peticiones se han usado para consultas, las cuales no alteran el estado de la base de datos. Esta petición, dentro del controlador de Spring Boot, se realiza mediante la notación @GetMapping. Estas peticiones pueden contener información mediante parámetros en la URL.

```
@GetMapping("/getAll")
public List<Usuario> getAllUsers(){
    List<Usuario> listaUsuarios = usuarioService.findAll();
    for(int i = 0; i < listaUsuarios.size(); i++) {
        listaUsuarios.get(i).setPassword("");
    }

    return listaUsuarios;
}
```

4. Ejemplo de notación GetMapping

- **POST**: las peticiones de tipo POST, son utilizadas para cuando se quiere **enviar información de gran tamaño** (imágenes, paquetes de datos) o información de carácter privado. Esta petición dentro del controlador de Spring Boot se realiza mediante la notación @PostMapping. La información, al contrario que con las de tipo GET, se adjunta dentro del encabezado HTTP.

```java
@PostMapping("/login")
public Map<String, Object> getByUsername(@RequestBody Map<String, String>
    Usuario usuario = usuarioService.findUserLogin(json.get("username"));
    Map<String, Object> result = new HashMap<>();

    if(usuario != null) {
        if(usuario.getPassword().equals(json.get("password"))) {
            result.put("code", "ok");
            result.put("user", usuario);
        }
        else {
            result.put("code", "password");
            result.put("error", "Contraseña incorrecta");
        }
    }
    else {
        result.put("code", "username");
        result.put("error", "Usuario no encontrado en la aplicacion");
    }

    return result;
}
```

5. Ejemplo de notación PostMapping

- **DELETE**: las peticiones de tipo DELETE, son utilizadas para **borrar un recurso o dato específico** y como la petición de tipo GET, puede contener parámetros dentro de la URL. Esta petición en Spring Boot se realiza mediante la notación @DeleteMapping.

```java
@DeleteMapping("/unfavBook")
@ResponseStatus(HttpStatus.NO_CONTENT)
public void unfavBook(@RequestBody Map<String, String> json){
    ListaLibroId listaId = new ListaLibroId();
    listaId.setLibro(Long.parseLong(json.get("item")));
    listaId.setUsuario(Long.parseLong(json.get("user")));

    listaService.libroDelete(listaId);
}
```

6. Ejemplo de notación DeleteMapping

- **PUT**: las peticiones de tipo PUT, son utilizadas para realizar un **reemplazo de un recurso o datos**, siendo similar a una sentencia de tipo UPDATE y la información de la petición, al igual que en la petición POST, se adjunta dentro del encabezado HTTP. Esta petición en Spring Boot se realizar mediante la notación @PutMapping.

```java
@PutMapping("/update/{id}")
@ResponseStatus(HttpStatus.CREATED)
public Map<String, Object> updateUser(@PathVariable Long id, @RequestBody Map<String, Str
    Usuario usuario = usuarioService.findById(id);

    usuario.setNombre(json.get("nombre"));
    usuario.setApellidos(json.get("apellidos"));
    usuario.setEmail(json.get("email"));
    usuario.setUsername(json.get("username"));
    if(!json.get("fecha").equals("")) usuario.setCumpleanos(new SimpleDateFormat("yyyy-mm
    else usuario.setCumpleanos(null);
    if(!json.get("passwd").equals("")) usuario.setPassword(json.get("passwd"));

    Usuario newUsuario = usuarioService.save(usuario);
    Map<String, Object> result = new HashMap<>();

    if(newUsuario != null) {
        result.put("code", "ok");
        result.put("user", newUsuario);
    }
    else {
        result.put("code", "error");
        result.put("error", "No se ha podido actualizar los datos del usuario.");
    }

    return result;
}
```

7. Ejemplo de notación PutMapping

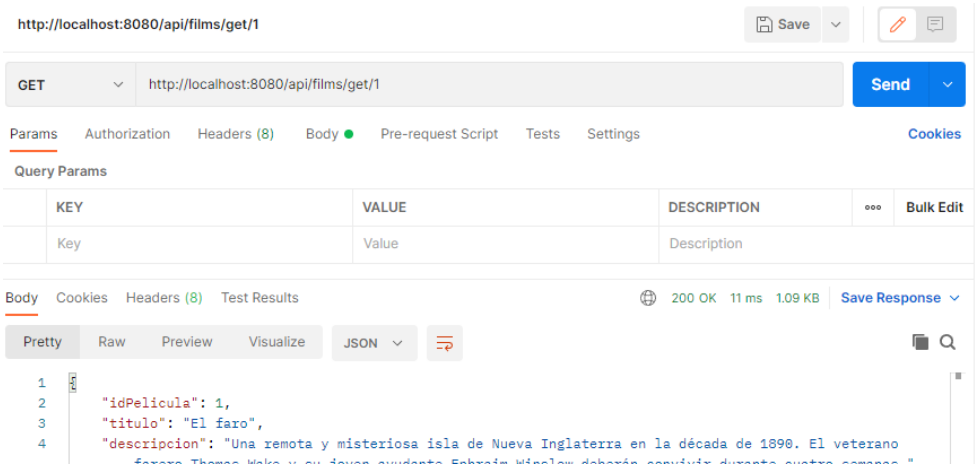
Una herramienta que ha sido de gran ayuda para poder probar el correcto funcionamiento del API REST, mediante peticiones HTTP, ha sido **Postman**, la cual se explicará en más detalle en el siguiente apartado.

3.2.2. Postman

Como se ha mencionado en el anterior apartado, una parte del desarrollo del proyecto ha sido la creación de un API REST, por lo que ha sido bastante importante, poder **probar todas las peticiones y *endpoints*** que se han programado en el API.

Para realizar estas pruebas, se ha utilizado la herramienta Postman, que es una aplicación dirigida a desarrolladores web, que permite realizar peticiones HTTP a cualquier API, y poder así, comprobar el correcto funcionamiento de estas.

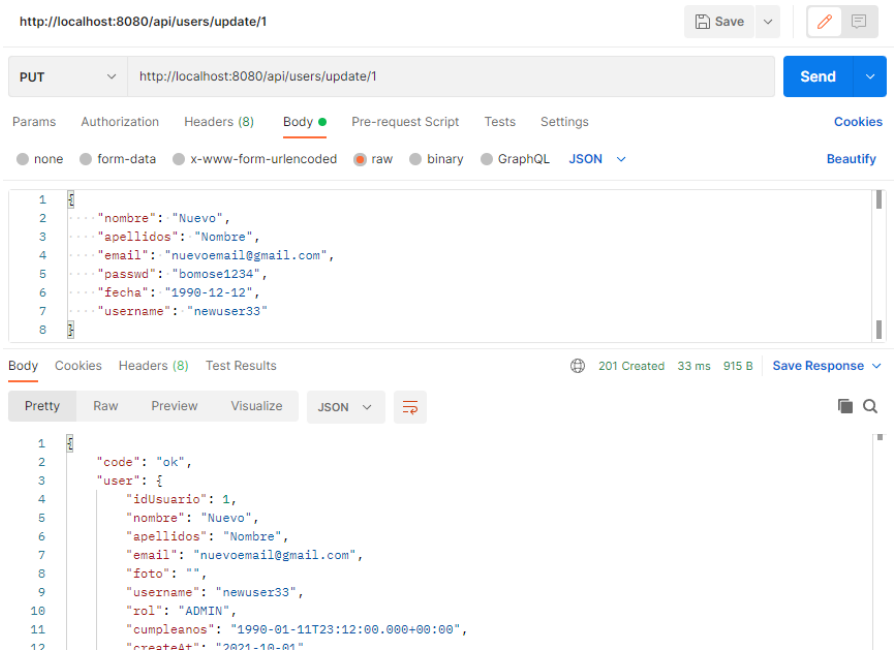
Además, Postman cuenta con una interfaz simple y una gran cantidad de funcionalidades. A continuación, se muestran distintas pruebas realizadas utilizando esta herramienta.



8. Ejemplo de petición GET con Postman


```
{
    "idPelicula": 1,
    "titulo": "El faro",
    "descripcion": "Una remota y misteriosa isla de Nueva Inglaterra en la década de 1890. El vete
    "duracion": 70,
    "foto": "films/el-faro.jpg",
    "links": "",
    "tags": "drama;terror",
    "fecha": "2018-01-10",
    "listaResenas": [],
    "listaFavoritos": [],
    "direccion": [
        {
            "autor": {
                "idAutor": 13,
                "nombre": "Robert",
                "apellidos": "Eggers"
            },
            "pelicula": {
                "idPelicula": 1,
                "titulo": "El faro",
                "foto": "films/el-faro.jpg"
            },
            "rol": "director"
        }
    ],
    "reparto": [
        {
            "actor": {
                "idActor": 1,
                "nombre": "Willem",
                "apellidos": "Dafoe"
            },
            "pelicula": {
                "idPelicula": 1,
                "titulo": "El faro",
                "foto": "films/el-faro.jpg"
            },
            "papel": "Thomas Wake"
        },
        {
            "actor": {
                "idActor": 2,
                "nombre": "Robert",
                "apellidos": "Pattinson"
            },
            "pelicula": {
                "idPelicula": 1,
                "titulo": "El faro",
                "foto": "films/el-faro.jpg"
            },
            "papel": "Thomas Howard"
        }
    ]
}
```

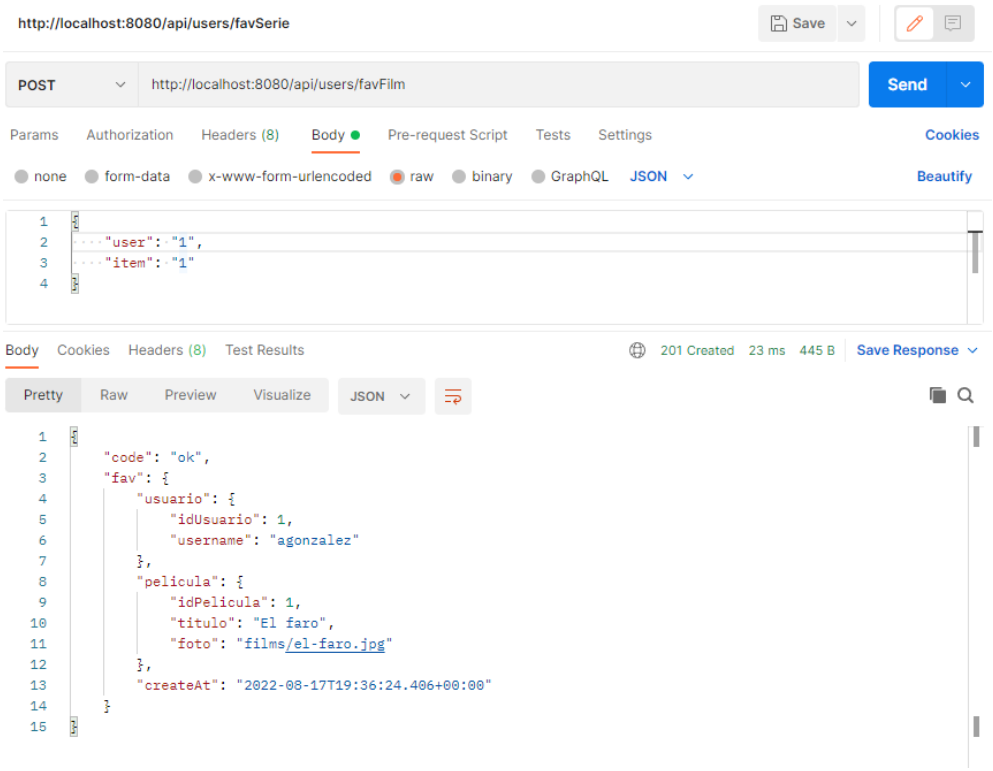

9. Resultado de la petición GET con Postman



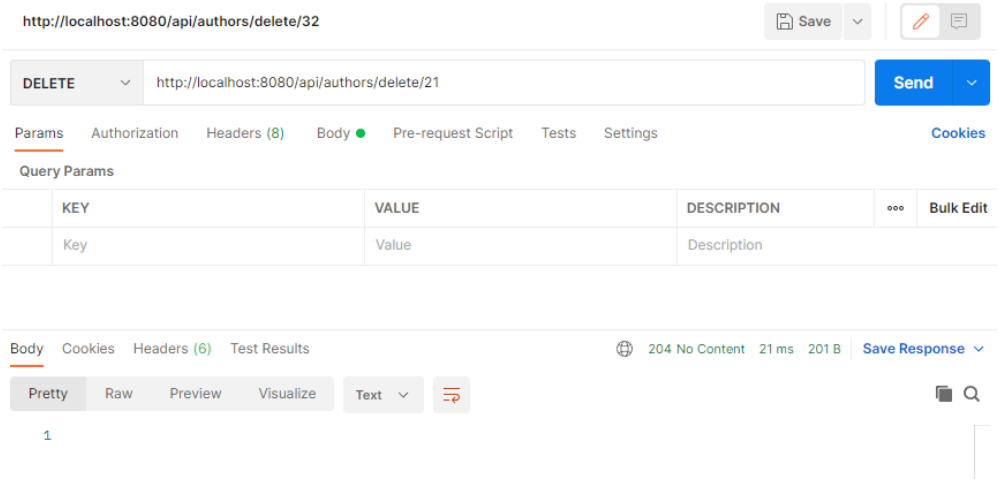
10. Ejemplo de petición PUT con Postman


```
{
    "code": "ok",
    "user": {
        "idUsuario": 1,
        "nombre": "Nuevo",
        "apellidos": "Nombre",
        "email": "nuevoemail@gmail.com",
        "foto": "",
        "username": "newuser33",
        "rol": "ADMIN",
        "cumpleanos": "1990-01-11T23:12:00.000+00:00",
        "createAt": "2021-10-01",
        "resenaLibros": [],
        "listaLibros": [],
        "resenaPeliculas": [],
        "listaPeliculas": [
            {
                "usuario": {
                    "idUsuario": 1,
                    "username": "newuser33"
                },
                "pelicula": {
                    "idPelicula": 1,
                    "titulo": "El faro",
                    "foto": "films/el-faro.jpg"
                },
                "createAt": "2022-08-17"
            }
        ],
        "resenaSeries": [],
        "listaSeries": [
            {
                "usuario": {
                    "idUsuario": 1,
                    "username": "newuser33"
                },
                "serie": {
                    "idSerie": 1,
                    "titulo": "Breaking Bad",
                    "foto": "series/breaking-bad.jpg"
                },
                "createAt": "2022-08-17"
            }
        ],
        "password": "bomose1234"
    }
}
```


11. Resultado de la petición PUT con Postman



12. Ejemplo de petición POST con Postman



13. Ejemplo de petición DELETE con Postman

3.2.3. ReactJS

Otra de las tecnologías más importantes y que mayor peso tiene en el desarrollo del proyecto es ReactJS, la cual es una **librería OS (Open Source)** lanzada en 2013 y desarrollada por Facebook, que utiliza el lenguaje de programación **Javascript** y permite desarrollar **aplicaciones front-end SPA (Single Page Application).**

Se ha elegido ReactJS por las siguientes razones:

- La aplicación generada por ReactJS, como se ha dicho anteriormente, es **Single Page Application**, es decir, la aplicación web únicamente consta de una página, la cual renderiza diferentes vistas. Esto es una ventaja ya que la navegación es mucho más rápida y brinda una mejor experiencia de usuario.
- Tiene **gran modularidad** ya que las diferentes vistas presentes en una aplicación de ReactJs, están compuestas, de lo que se denomina en ReactJS, componentes, los cuales son funciones o clases que renderizan código. Estos componentes son totalmente reutilizables y pueden personalizarse mediante el uso de *"props"*.
- Gracias al uso de componentes, el **mantenimiento** de la aplicación es mucho **más rápido**, simple, hace que la aplicación tenga una **gran escalabilidad** y **reduce la cantidad de código necesario**.
- Es una tecnología con **gran soporte** y una **gran cantidad de librerías** que hacen el desarrollo mucho más rápido y fácil, gracias a que, como se ha mencionado anteriormente, es una librería de código abierto.
- **Fácil de aprender**.
- Tiene **mayor compatibilidad con *SEO*** que sus competidores.

Algunas de las librerías usadas en la aplicación ReactJS son:

- sass (https://www.npmjs.com/package/sass)
- react-slick (https://www.npmjs.com/package/react-slick)
- react-router-dom (https://v5.reactrouter.com/web/guides/quick-start)
- react-helmet (https://www.npmjs.com/package/react-helmet)

- bootstrap-icons (https://www.npmjs.com/package/bootstrap-icons)

3.2.4. Bootstrap

Como se ha mencionado anteriormente, uno de los objetivos de este proyecto, es crear una página web con un diseño responsive, por eso, se ha optado por utilizar Bootstrap en este proyecto. Bootstrap es un ***framework* CSS** de código abierto, desarrollado inicialmente por Twitter en 2011.

Alguna de las ventajas que proporciona Bootstrap son:

- **Implementa un sistema GRID**, con el que se puede elegir el ancho y posición del contenido mediante columnas, todo ello ajustado al ancho de la pantalla del dispositivo. Este GRID divide la pantalla en 12 columnas pudiendo poner dentro de una misma fila (*row*) columnas que sumen un total de 12 unidades, usando la notación "col-2", "col-4" …
- **Clases con estilos predeterminados**, estas clases pueden poner **diferentes estilos**, tales como, *paddings, margins, display,* estilos de textos, colores…
- **Sistema de *Media Queries*** mediante clases, para diferentes tamaños de pantalla, lo que permite realizar la maquetación en diferentes dispositivos de una forma más fácil y rápida. Estas notaciones se pueden mezclar con otras funcionalidades de Bootstrap, como, por ejemplo; *paddings (p-md-2)*, *margins (my-sm-1)*, *displays (d-lg-flex)*, sistema GRID (col-xl-3) …

Breakpoint	Class infix	Dimensions
X-Small	*None*	<576px
Small	sm	≥576px
Medium	md	≥768px
Large	lg	≥992px
Extra large	xl	≥1200px
Extra extra large	xxl	≥1400px

14. Tabla de Media Queries Bootstrap

- **Facilita el diseño y la maquetación** debido a todas las herramientas y funcionalidades que se han expuesto anteriormente.
- Genera una **experiencia de usuario agradable y atractiva.**
- **Documentación extensa y completa.**
- Tiene **gran soporte y compatibilidad** con la mayoría de los navegadores del mercado.

4. Desarrollo de la aplicación

4.1. Base de datos y modelo entidad-relación

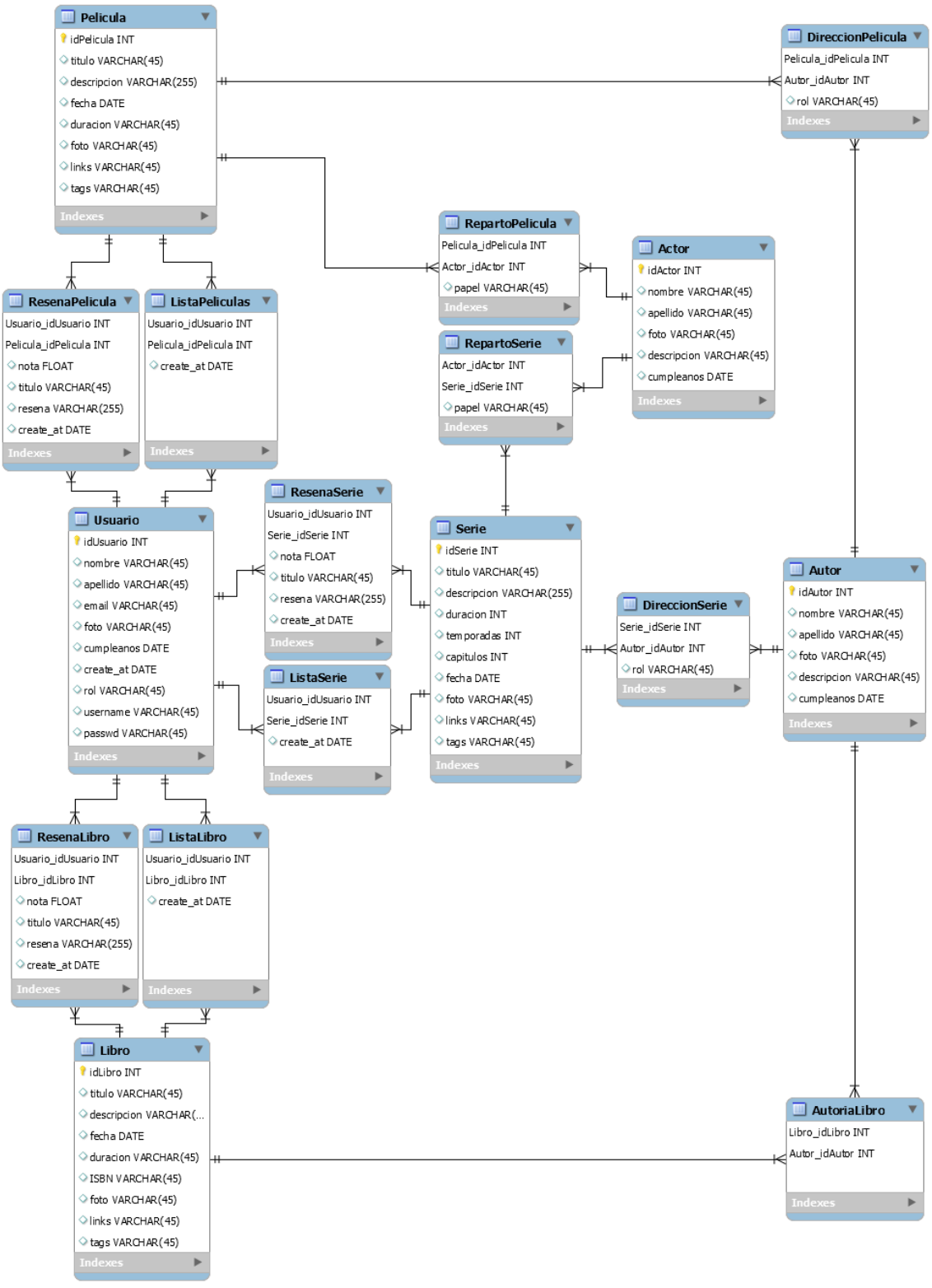
Cuando se empezó el desarrollo de la aplicación, una de las primeras tareas fue **planificar, elegir y diseñar la base de datos** que almacenaría toda la información referente a usuarios, películas, series, autores…

Para este desarrollo, se ha valorado usar una **base de datos relacional**, ya que, de esta forma, el acceso y escritura de la información es mucho más fácil que con una base de datos no relacional.

La base de datos usada en este caso ha sido MySQL, la cual se ha elegido por las siguientes razones:

- **Compatibilidad** con la tecnología **Spring Boot**, algo indispensable para el desarrollo del proyecto.
- Es **rápida**.
- Es **compatible** y se puede ejecutar en **múltiples sistemas operativos**.
- Es **fácil de usar** y gracias a que es una base de datos ampliamente utilizada, se puede encontrar ayuda en numerosos foros o tutoriales.
- El **coste** es **bastante inferior** al de la base de datos de la competencia, pero, para este proyecto, debido a que es un entorno de pruebas, el coste es gratuito.
- Es **escalable**.
- Muchos de los **servicios de *hosting*** tienen MySQL como opción a la hora de elegir una base de datos.

El modelo entidad-relación que se ha diseñado es el siguiente:



15. Modelo Entidad-Relación

Las entidades principales que se encuentran en este modelo son:

- **Usuario**: tabla con la información de los **usuarios** que usan la aplicación. Tiene una relación de muchos a muchos con serie, película y libro, para almacenar las reseñas y las listas de favoritos.
- **Libro**: tabla con la información de todos los **libros** que son accesibles desde la aplicación. Tiene una relación de muchos a muchos con usuario y autor, para almacenar las reseñas, listas de favoritos y autoría del libro.
- **Serie**: tabla con la información de todas las **series** que son accesibles desde la aplicación. Tiene una relación de muchos a muchos con usuario, autor y actor, para almacenar las reseñas, listas de favoritos, dirección y reparto de la serie.
- **Película**: tabla con la información de todas las **películas** que son accesibles desde la aplicación. Tiene una relación de muchos a muchos con usuario, autor y actor, para almacenar las reseñas, listas de favoritos, dirección y reparto de la película.
- **Autor**: tabla con la información de todos los **autores** que son accesibles desde la aplicación. Tiene una relación de muchos a muchos con serie, película y libro, para almacenar la información sobre la dirección o autoría de las obras.
- **Actor**: tabla con la información de todos los **actores** que son accesibles desde la aplicación. Tiene una relación de muchos a muchos con serie y película, para almacenar la información sobre el reparto de series y películas.

Como hemos dicho anteriormente, estas entidades tienen **relaciones de muchos a muchos**, las cuales generan **tablas intermedias** que almacenan los datos de estas relaciones. Estas tablas son las siguiente:

- **ResenaPelicula, ResenaLibro, ResenaSerie:** tablas que guardan la información sobre las **reseñas** que hacen los **usuarios** de los libros, series y películas dentro de la aplicación. Tiene información sobre la nota, título, texto y fecha de creación de la reseña.

- **ListaPelicula, ListaLibro, ListaSerie:** tablas que guardan la información sobre que películas, series o libros están marcados como favoritos por los usuarios de la aplicación. Tiene información sobre la fecha en la que se ha añadido la entrada en la lista.
- **RepartoPelicula, RepartoSerie:** tablas que guardan la información sobre que actores participan en las diferentes películas o series de la aplicación. Tiene información sobre el papel que tiene el actor dentro de la película o serie.
- **DireccionPelicula, DireccionSerie:** tablas que guardan la información sobre que autores dirigen en las diferentes películas o series de la aplicación. Tiene información sobre el rol que tiene el director dentro de la película o la serie.
- **AutoriaLibro:** tabla que guarda la información sobre que autores han escrito los diferentes libros de la aplicación.

Este modelo entidad-relación no se ha implementado directamente mediante un *script* SQL en MySQL, si no que, como se ha explicado en apartados anteriores, el modelo se representa en Spring Boot mediante clases de Java y distintas notaciones para representar nombre de las tablas, relaciones, nombre de atributos, y, posteriormente**, Spring Boot** se encarga de **crear todas las tablas** y **relaciones** dentro de la base de datos de **forma automática.**

4.2. Requisitos

En este apartado, se realizará un análisis de todos los requisitos, funcionales y no funcionales, para conocer los elementos necesarios a la hora de desarrollar el proyecto.

4.2.1. Requisitos funcionales

Identificativo	Descripción
RF-01	Permitir a los usuarios crear una cuenta en la aplicación.
RF-02	Permitir a los usuarios con permisos de administrador, crear una cuenta a otro usuario, con la posibilidad de hacer dar el rol de administrador al nuevo usuario.
RF-03	Permitir a los usuarios con permisos de administrador, editar la información de la cuenta de un usuario.
RF-04	Diferentes funcionalidades y vistas en función del nivel de permisos que tenga el usuario.
RF-05	Permitir a los usuarios cambiar la información de su perfil.
RF-06	Mostrar un listado que reúna todos los libros, películas, series, actores y autores de la aplicación.
RF-07	Ordenar los libros, películas, series, actores y autores en función del nombre, fecha, nota media o id.
RF-08	Mostrar la información, reparto y dirección de los libros, películas, series.
RF-09	Mostrar la información y obras escritas, dirigidas o en las que aparecen los actores y autores.
RF-10	Mostrar el perfil de los usuarios de la aplicación con su información y reseñas.
RF-11	Permitir a los usuarios con una cuenta creada, valorar y reseñar los libros, películas y series de la aplicación.
RF-12	Permitir a todos los usuarios de la aplicación, ver las reseñas y valoraciones creadas por otros usuarios.
RF-13	Permitir a los usuarios con una cuenta creada, añadir y/o eliminar de su lista de favoritos los libros, películas y series de la aplicación.
RF-14	Controlar que los usuarios solo puedan hacer una reseña por cada libro, película o serie.

RF-15	Permitir únicamente a los usuarios con rol de administrador editar la información referente a películas, series, libros, actores y autores.
RF-16	Permitir únicamente a los usuarios con rol de administrador borrar y crear nuevas entradas de películas, series, libros, actores y autores.

4.2.2. Requisitos no funcionales

4.2.2.1. Hardware y software

Identificativo	Descripción
RNF-01	Los datos de los usuarios, reseñas, favoritos, películas, series, actores y autores serán almacenados en una base de datos MySQL.
RNF-02	Se desarrollará un API REST utilizando el lenguaje de programación Java, mediante el *framewok* Spring Boot usando el IDE Spring Tool Suite 4, que sirva para comunicar la aplicación web y la base de datos entre sí.
RNF-03	La aplicación web será desarrollada con ReactJS usando el editor de código Visual Studio Code.
RNF-04	La aplicación web es compatible con los navegadores más usados actualmente.

4.2.2.2. Calidad del sistema

Identificativo	Descripción
RNF-05	La aplicación implementará validación para los formularios en los que el usuario tenga que introducir o editar datos.
RNF-06	Se utilizará el patrón de diseño MVC (Modelo Vista Controlador).
RNF-07	Se utilizarán tecnologías y lenguajes ampliamente usados, con gran soporte y actualizaciones.

4.2.2.3. Usabilidad del sistema

Identificativo	Descripción
RNF-08	El diseño de la interfaz será amigable para el usuario.
RNF-09	La página web será completamente navegable desde cualquier dispositivo, independientemente de las dimensiones de este

4.2.2.4. Seguridad

Identificativo	Descripción
RNF-10	El acceso a algunas vistas, contenido y funcionalidad de la web estará restringido para usuarios no registrados en la aplicación.
RNF-11	Las funciones críticas solo estarán disponibles para los usuarios que posean el rol de administrador.
RNF-12	Las comunicaciones con el API que contengan datos críticos se realizarán mediante peticiones de tipo POST para no comprometer los datos.

4.3. Diagrama de casos de uso

El diagrama de casos de uso ayuda a especificar de una forma gráfica, los requisitos funcionales que el sistema debe cumplir y como este interactúa con el usuario para conseguirlo.

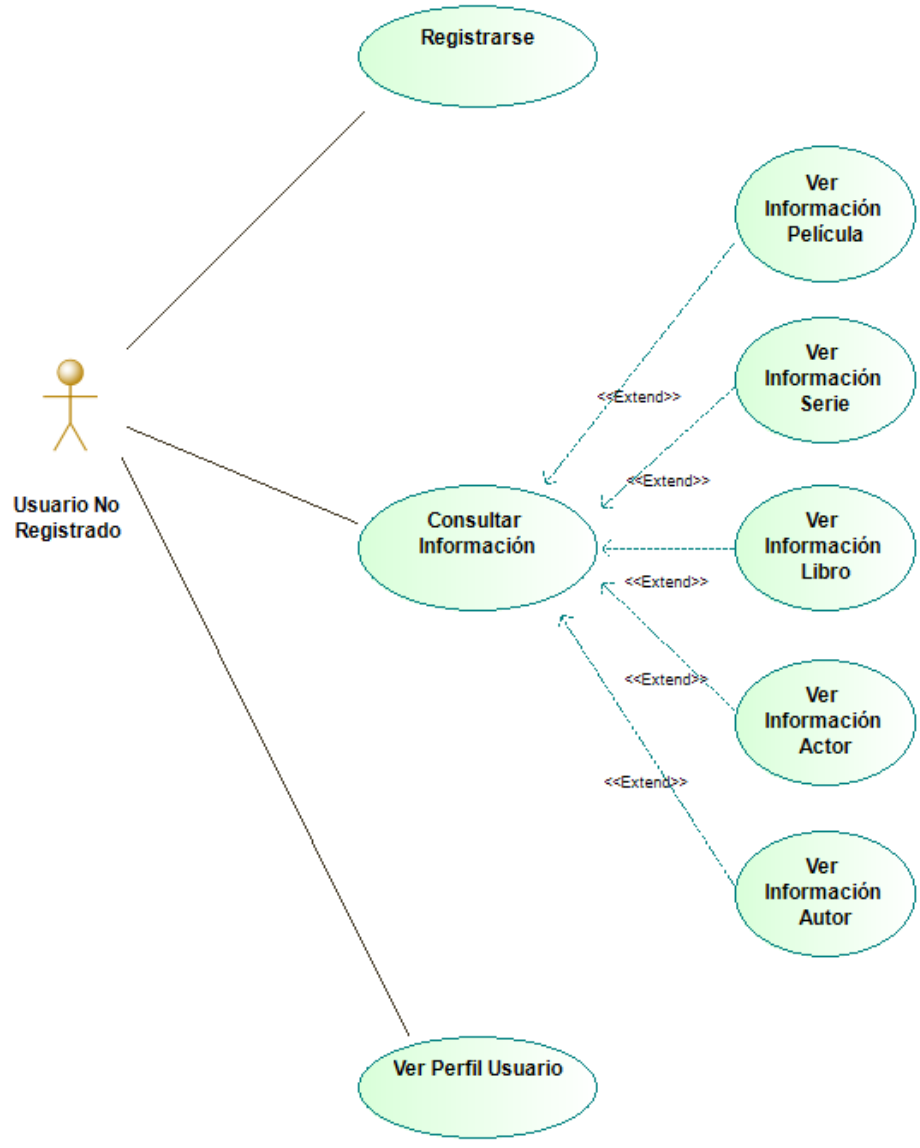
Los elementos fundamentales que se observan en este diagrama son los siguientes:

- **Actor:** es representado mediante el icono de una persona. Representa una persona o grupo de personas que cumplen un papel en la interacción con el sistema.
- **Caso de uso:** es representado mediante un círculo u óvalo con el nombre del caso de uso en su interior. Representa una funcionalidad o requisito del sistema.

- **Relación:** es representado mediante una flecha que une dos casos de uso o un caso de uso y un actor. Representa las interacciones que se producen en el sistema.

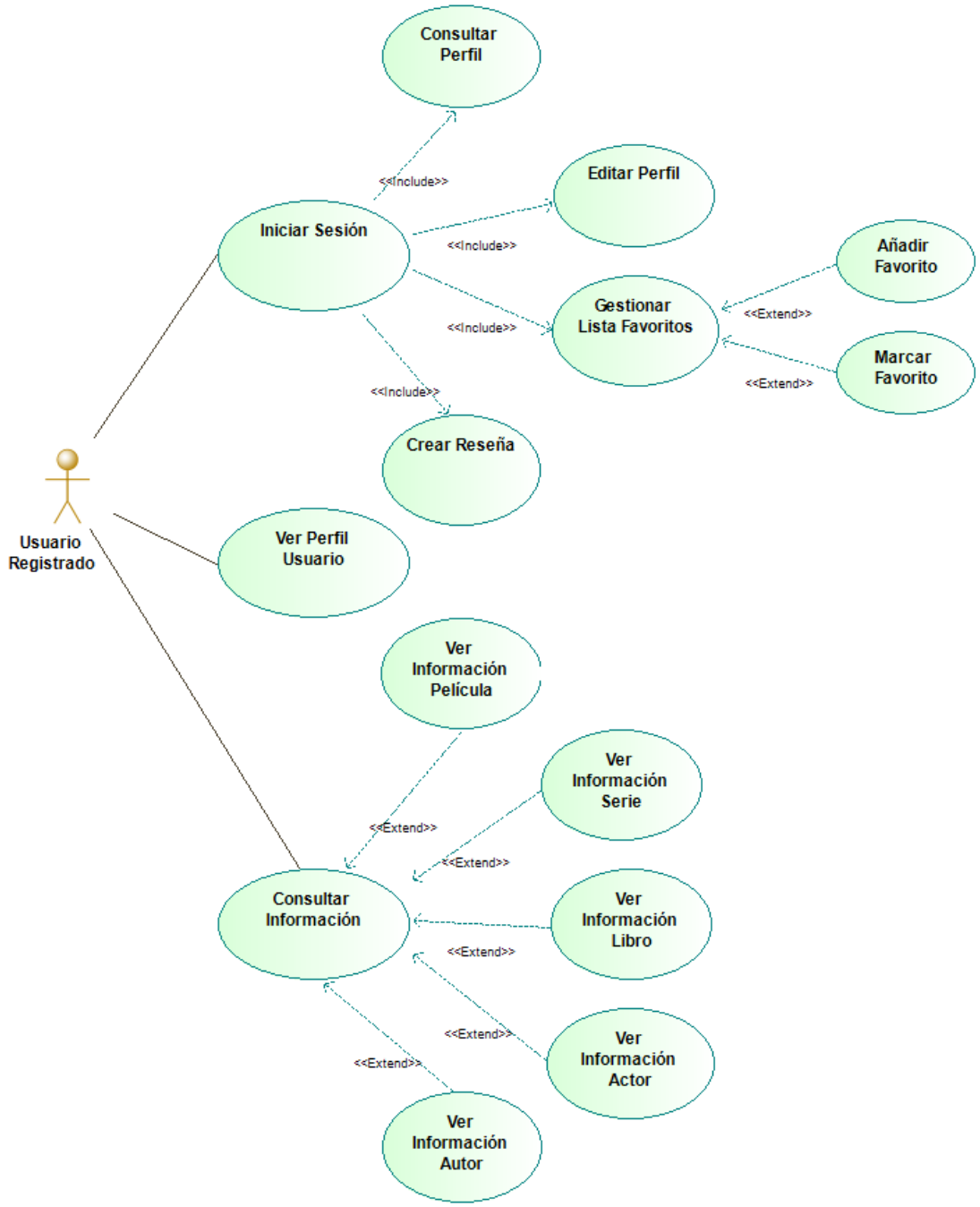
En este caso, se realizarán tres casos de uso, según el tipo de usuario que interaccione con el sistema. Los tipos de usuario que podrán interaccionar con el sistema serán**; usuarios registrados, usuarios no registrados, administradores.**

4.3.1. Usuario no registrado



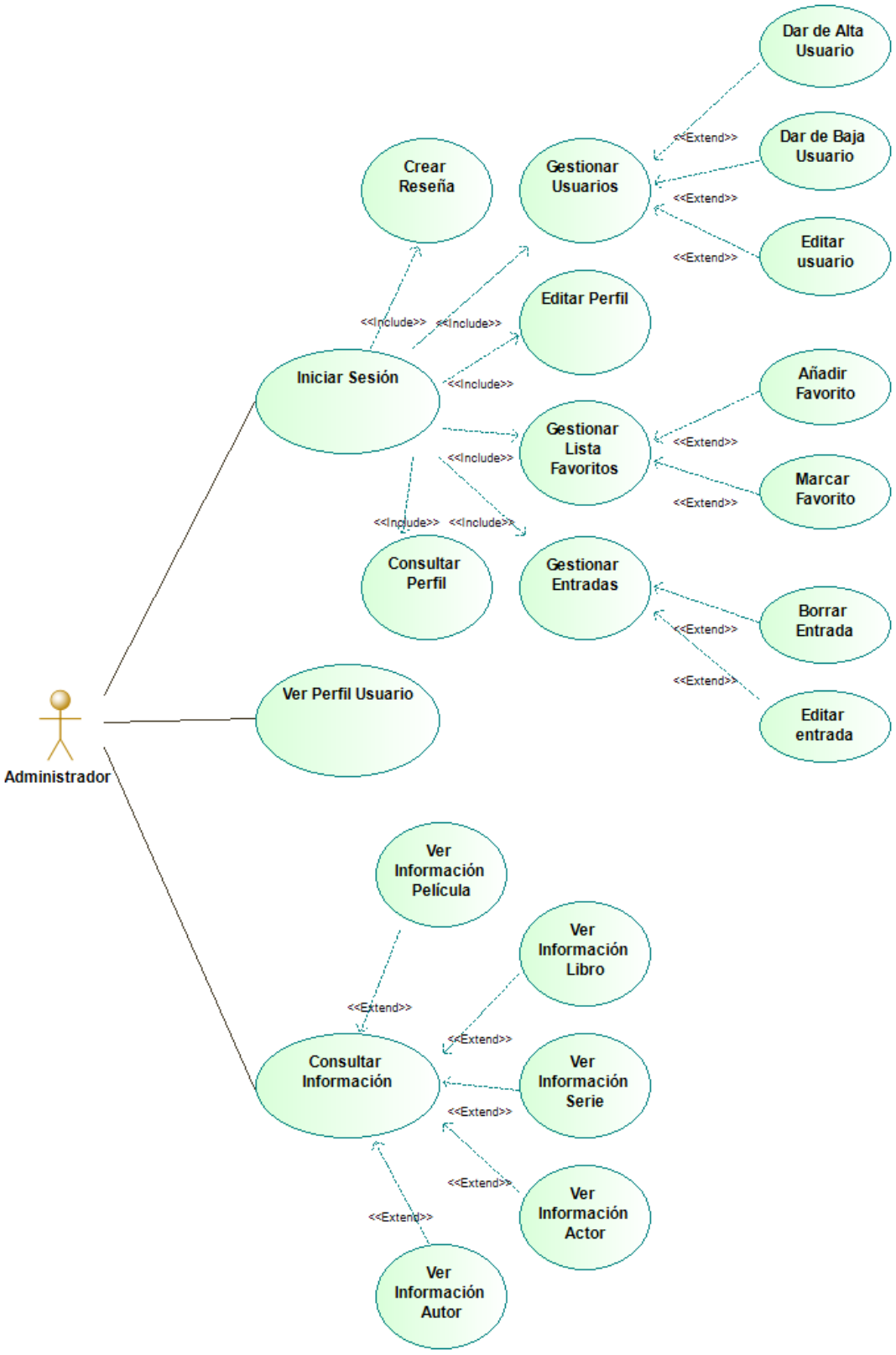
16. Caso de uso Usuario no registrado

4.3.2. Usuario registrado



17. Caso de uso Usuario registrado

4.3.3. Administrador



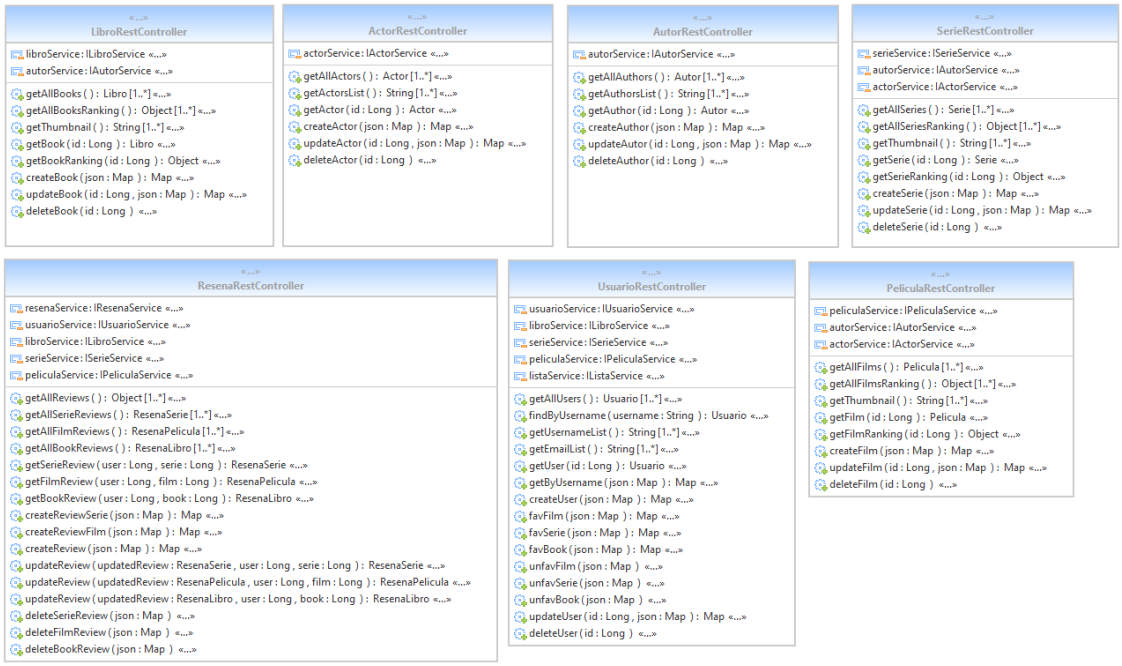
18. Caso de uso Administrador

4.4. Diagrama de clases UML

Para este apartado se ha utilizado la herramienta UML Lab Modelling, la cual se ha instalado desde el buscador de extensiones y plugins que tiene Eclipse. Esta herramienta ayuda a generar, de forma automática, diagramas UML a partir de clases Java.

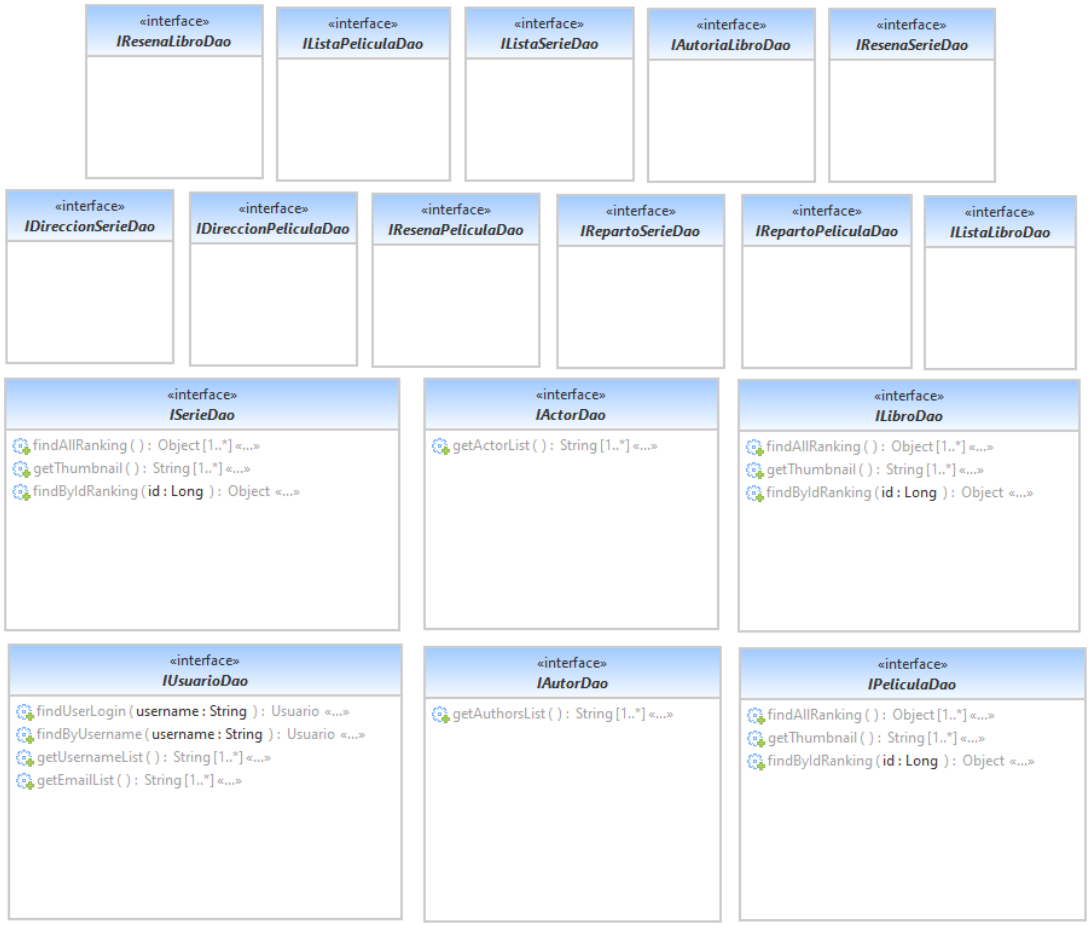
Los diagramas UML serán divididos por los paquetes que hay en la aplicación de Spring Boot, por ello, muchas de las relaciones o usos

4.4.1. Paquete Controllers



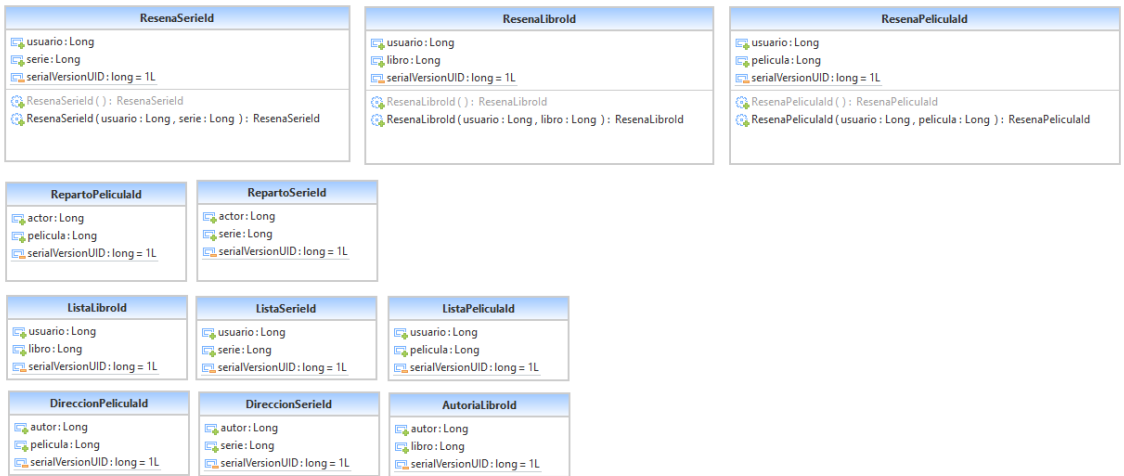
19. Diagrama UML paquete "Controllers"

4.4.2. Paquete Dao



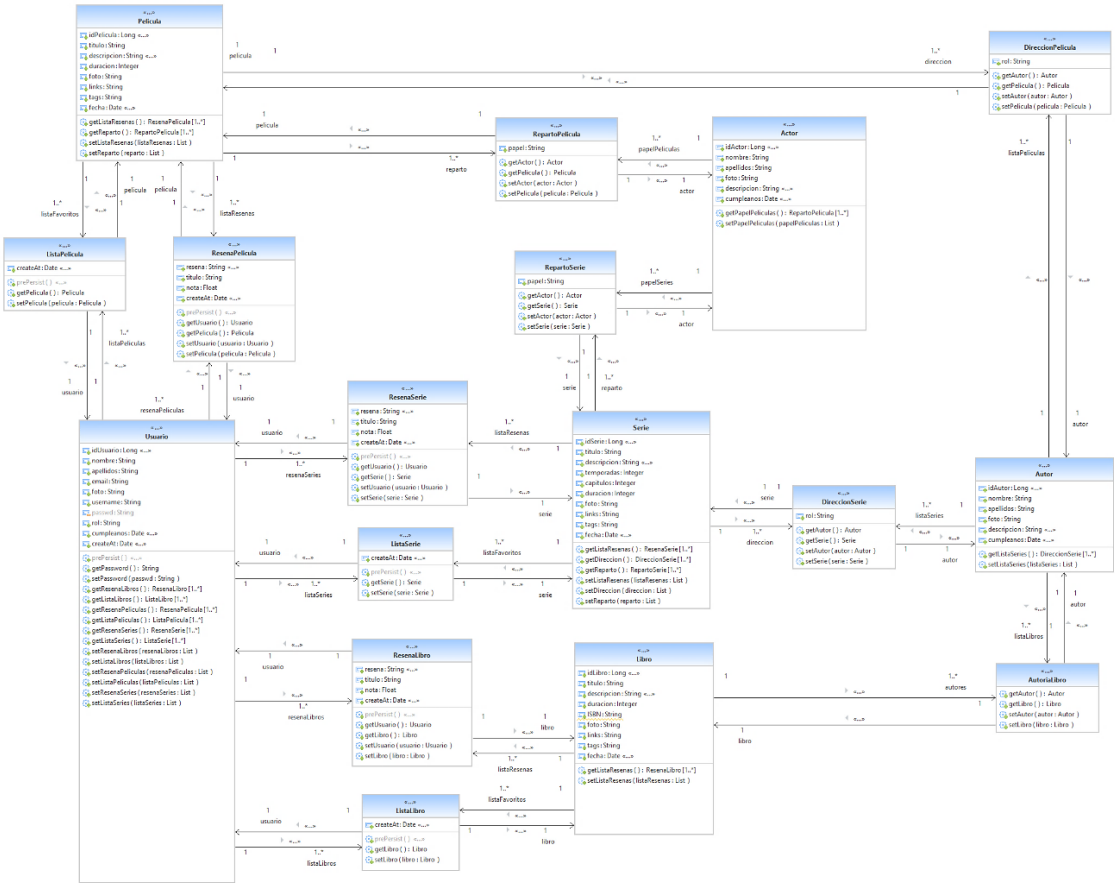
20. Diagrama UML paquete "Dao"

4.4.3. Paquete Dao.FK



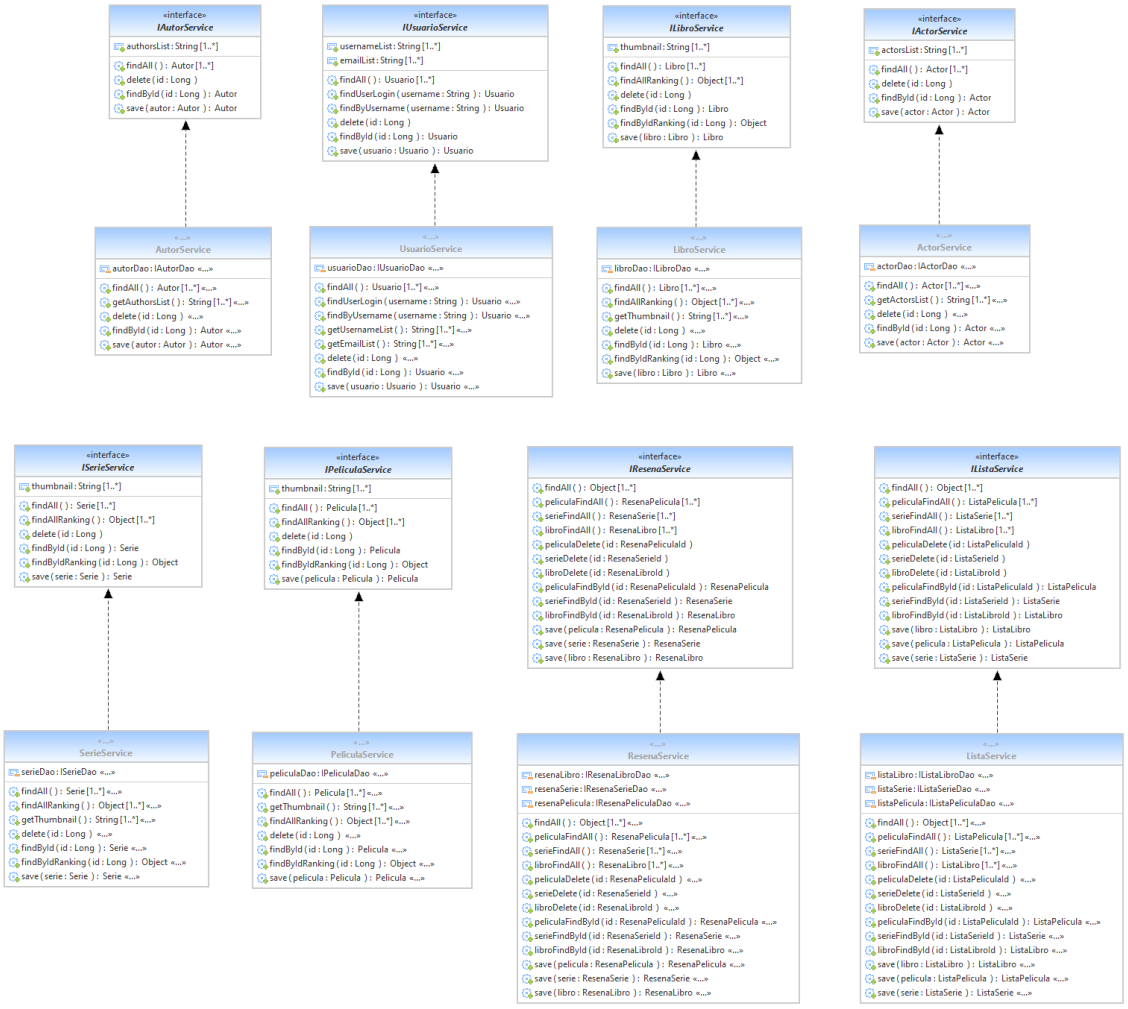
21. Diagrama UML paquete "Dao.FK"

4.4.4. Paquete Entity



22. Diagrama UML paquete "Entity"

4.4.5. Paquete Service

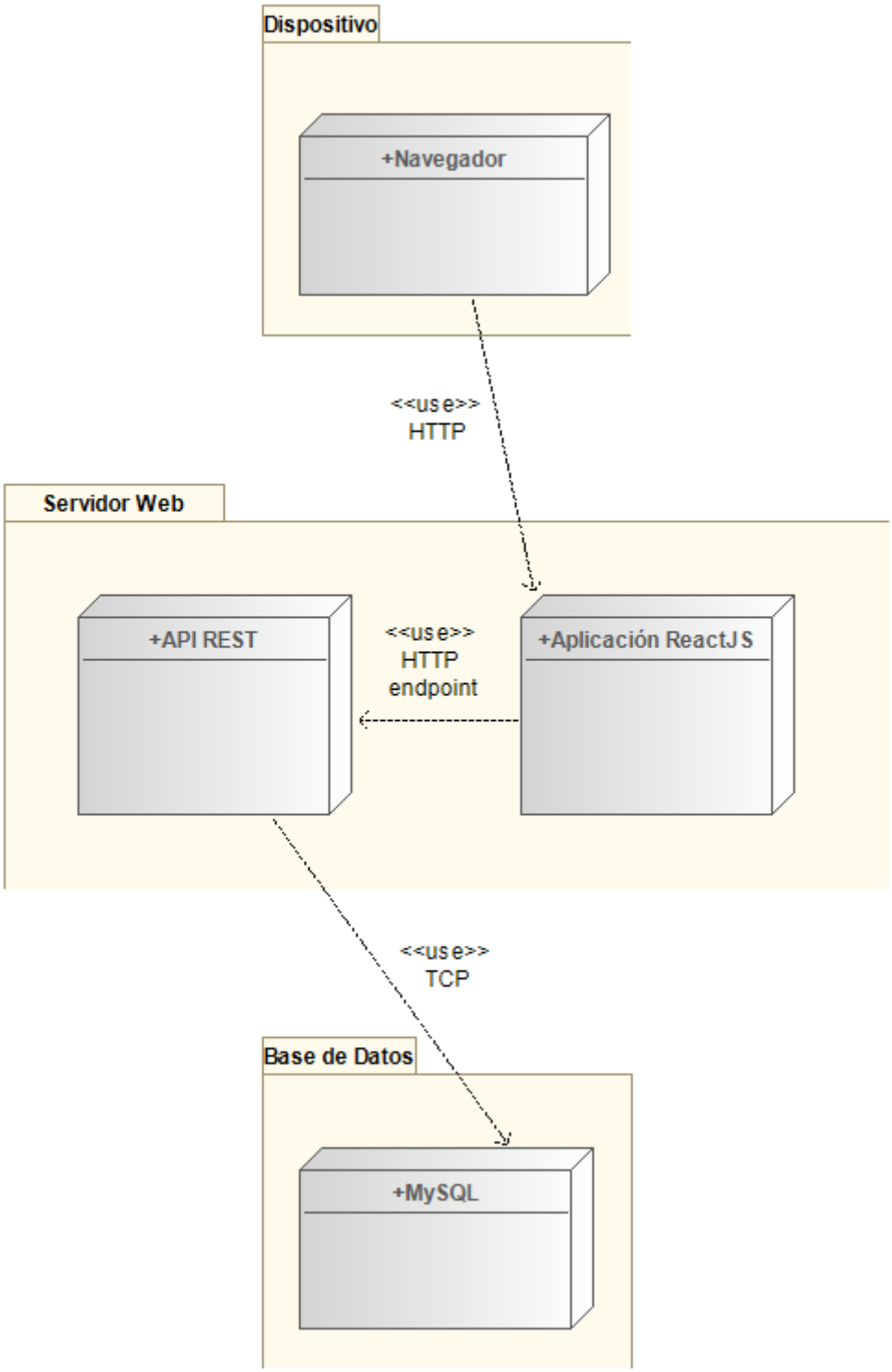


23. Diagrama UML paquete "Service"

4.5. Diagrama de despliegue

El proyecto desarrollado, como se ha explicado con anterioridad, es una aplicación full-stack, y mediante un diagrama de despliegue se muestra cual es la **estructura general del proyecto**, representando las diferentes partes que lo componen mediante **nodos.**

Los nodos representados en este diagrama son; el **dispositivo del usuario**, aplicación **o interfaz ReactJS**, el **API REST** de Spring Boot y la **base de datos** MySQL.



24. Diagrama de despliegue

4.6. Código de la aplicación

En este apartado se explicarán y mostrarán los fragmentos de código más importantes o interesantes de la aplicación.

4.6.1. Controladores del API REST

Una de las **partes** más **esenciales del desarrollo** es, que los controladores del API REST tengan un correcto funcionamiento y provean de todas las funcionalidades necesarias al usuario y a la aplicación.

A continuación, se mostrará el código principal y algunas de las funciones del **controlador de actores** (ActorRestController).

```java
package com.springboot.api.rest.bomose.controllers;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import com.springboot.api.rest.bomose.models.entity.Actor;
import com.springboot.api.rest.bomose.models.service.IActorService;

@CrossOrigin(origins = {"http://localhost:8080", "http://localhost:3000"})
@RestController
@RequestMapping("/api/actors")
public class ActorRestController {

    @Autowired
    private IActorService actorService;

```

25. Código principal clase ActorRestController

En la imagen anterior se pueden ver cómo, mediante la notación **@CrossOrigin**, se indica al controlador, que **URL pueden acceder a él**.

Por otro lado, con la notación **@RequestMapping**, se indica cual es **la ruta del *endpoint*** para este controlador, es decir, la URL a la que hay que acceder para realizar peticiones.

Por último, se importa, en este caso, la clase IActorService mediante la notación @Autowired, que es una **interfaz** que contiene todas las **operaciones CRUD** y diferentes **funcionalidades** o **consultas** a la base de datos.

```
42
43      @GetMapping("/get/{id}")
44      public Actor getActor(@PathVariable Long id) {
45          return actorService.findById(id);
46      }
47
```

26. Método GetMapping del controlador

```
47
48      @PostMapping("/create")
49      @ResponseStatus(HttpStatus.CREATED)
50      public Map<String, Object> createActor(@RequestBody Map<String, String> json) throws ParseExcep
51          Actor actor = new Actor();
52
53          actor.setNombre(json.get("nombre"));
54          actor.setApellidos(json.get("apellidos"));
55          actor.setDescripcion(json.get("descripcion"));
56          actor.setFoto(json.get("foto"));
57          if(!json.get("fecha").equals("")) actor.setCumpleanos(new SimpleDateFormat("yyyy-mm-dd").pa
58          else actor.setCumpleanos(null);
59
60          Map<String, Object> result = new HashMap<>();
61          Actor newActor = actorService.save(actor);
62
63          if(newActor != null) {
64              result.put("code", "ok");
65              result.put("item", newActor);
66          }
67          else {
68              result.put("code", "save");
69              result.put("error", "No se ha podido crear la nueva entrada de actor.");
70          }
71
72          return result;
73      }
74
```

27. Método PostMapping del controlador

```
74
75      @PutMapping("/update/{id}")
76      @ResponseStatus(HttpStatus.CREATED)
77      public Map<String, Object> updateActor(@PathVariable Long id, @RequestBody Map<String, String>
78          Actor actor = actorService.findById(id);
79
80          actor.setNombre(json.get("nombre"));
81          actor.setApellidos(json.get("apellidos"));
82          actor.setDescripcion(json.get("descripcion"));
83          actor.setFoto(json.get("foto"));
84          if(!json.get("fecha").equals("")) actor.setCumpleanos(new SimpleDateFormat("yyyy-mm-dd").pa
85          else actor.setCumpleanos(null);
86
87          Map<String, Object> result = new HashMap<>();
88          Actor newActor = actorService.save(actor);
89
90          if(newActor != null) {
91              result.put("code", "ok");
92              result.put("item", newActor);
93          }
94          else {
95              result.put("code", "save");
96              result.put("error", "No se ha podido guardar el actor.");
97          }
98
99          return result;
100     }
101
```

28. Método PutMapping del controlador

```
101
102     @DeleteMapping("/delete/{id}")
103     @ResponseStatus(HttpStatus.NO_CONTENT)
104     public void deleteActor(@PathVariable Long id){
105         actorService.delete(id);
106     }
107 }
108
```

29. Método DeleteMapping del controlador

Las cuatro imágenes anteriores, muestran los cuatro tipos de funciones diferentes que puede tener un API en Spring Boot, cada una, representa un **tipo de petición HTTP** tal y como se ha explicado en el apartado 3.2.1. Spring Boot.

Estas funciones, pueden **recibir parámetros**, tanto en la URL del endpoint, usando la notación @PathVariable, como en el cuerpo de la petición HTTP, usando la notación @RequestBody.

4.6.2. Consulta sobre la nota media

Cuando se estaba desarrollando la aplicación, surgió la necesidad de tener en una **misma consulta a la base de datos**, la **información** referente a una película, serie o libro, **junto con la nota media de esta.**

Esto se consiguió mediante la notación **@Query**, la cual permite realizar **consultas personalizadas** utilizando el lenguaje SQL. Las consultas son las siguientes:

```
11
12      @Query("SELECT SUM(r.nota) / COUNT(*) AS nota, p FROM Pelicula"+
13      " p LEFT JOIN ResenaPelicula r ON r.pelicula.idPelicula=p.idPelicula"+
14      " GROUP BY p.idPelicula ORDER BY nota DESC")
15      public List<Object> findAllRanking();
16
```

30. Consulta SQL para la nota media de todas las películas

```
SELECT SUM(r.nota) / COUNT(*) AS nota, p FROM Pelicula p LEFT JOIN ResenaPelicula r ON r.pelicula.idPelicula=p.idPelicula GROUP BY p.idPelicula ORDER BY nota DESC
```

```
16
17      @Query("SELECT SUM(r.nota) / COUNT(*) AS nota, p FROM Pelicula"+
18      " p LEFT JOIN ResenaPelicula r ON r.pelicula.idPelicula=p.idPelicula"+
19      " WHERE p.idPelicula = ?1 GROUP BY p.idPelicula")
20      public Object findByIdRanking(Long id);
21
```

31. Consulta SQL para la nota media de una película

```
SELECT SUM(r.nota) / COUNT(*) AS nota, p FROM Pelicula p LEFT JOIN ResenaPelicula r ON r.pelicula.idPelicula=p.idPelicula WHERE p.idPelicula = ?1 GROUP BY p.idPelicula
```

4.6.3. Llamadas al API REST

Otra de las partes más importantes de la aplicación, en la parte front-end, es la **llamada a los diferentes endpoints del API REST** para **extraer la información** necesaria y poder **mostrársela al usuario.**

Para obtener la información, se utiliza la **interfaz *fecth()*.** Mediante esta interfaz, podemos especificar toda la información necesaria para realizar una petición HTTP, en este caso, al API REST.

A continuación, se muestra el código de una petición de tipo **GET** y **POST**.

```
fetch("http://localhost:8080/api/"+ type +"/get/"+ this.params.id)
    .then(response => {
        if(!response.ok) {
            this.setState({
                error: response.statusText
            });
        }

        return response.json();
    })
    .then((result) => {
        this.setState({
            item: result,
            isLoaded: true
        });

        if(type === "authors"){
            if(result["listaLibros"].length > 0){
                this.setState({rol: 'writer'});
            }
            else{
                this.setState({rol: 'director'});
            }

        }
        else if(type === "actors"){
            this.setState({rol: 'actor'});
        }

        document.title = "Bomose | "+ result["nombre"] +" "+ result["apellidos"]
        document.body.scrollTop = document.documentElement.scrollTop = 0;
    })
    .catch((error) => {
        this.setState({
            error: error.toString(),
            isLoaded: true
        });
    })
}
```

32. Llamada de tipo GET al API REST

```
const requestOptions = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      item: itemId,
      user: userId
    })
};

fetch("http://localhost:8080/api/users/"+endpoint, requestOptions)
    .then(response => response.json())
    .then((result) => {
        if(result["code"] === "ok"){
            this.setState({isFaved: true})
        }
        else{
            console.log(result["error"])
        }
    })
    .catch((error) => {

    })
}
```

33. Llamada de tipo POST al API REST

4.6.4. Implementación de la librería react-router-dom

Como se ha explicado en anteriores apartados, ReactJS, es un framework que genera una SPA (Single Page Aplication), esto tiene numerosas ventajas, pero a la hora de crear una web que tenga diferentes vistas, es decir, que **necesite ser navegable**, esto es un **problema**, ya que necesitamos algún **elemento** que **gestione la paginación.**

Por ello, se ha optado por implementar la librería, **react-router-dom**, la cual **gestiona** de **forma automática** el **cambio entre las diferentes vistas** de la aplicación, tratando cada una de las vistas, como un componente el cual se renderiza o no, dependiendo de la URL.

En el archivo index.js de la aplicación, se usa la etiqueta *BrowserRouter*, que es, una interfaz común de alto nivel para todos los componentes del router.

Dentro de esta etiqueta, se encuentra toda la lógica de vistas del router, dentro del archivo App.js.

```
import React from 'react';
import ReactDOM from 'react-dom/client';

import { BrowserRouter } from "react-router-dom";
import { RouterApp } from 'App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <BrowserRouter>
    <RouterApp></RouterApp>
  </BrowserRouter>
);
```

34. Implementación de react-router-dom en App.js

En el archivo App.js, todas las vistas se encuentren dentro de la etiqueta *Routes*. Estas vistas se indexan, mediante la etiqueta *Route*, la cual tiene los siguientes atributos:

- **path**: es la URL que mostrará la vista. Esta URL puede tener parámetros que se indican mediante dos puntos, por ejemplo, “/films/update/:id”, donde id es el parámetro, el cual podrá ser usado en la vista como un atributo más del componente.

- **element**: es la vista que se va a renderizar, en este caso, para alguna de las rutas, se ha usado renderizado condicional, en función de si el usuario es administrador o está registrado o no, para restringir el acceso a algunas de las vistas. La redirección se realiza con la etiqueta *Navigate.*

35. Código de implementación de las rutas del router pt.1

36. Código de implementación de las rutas del router pt.2

4.6.5. Banner personalizable

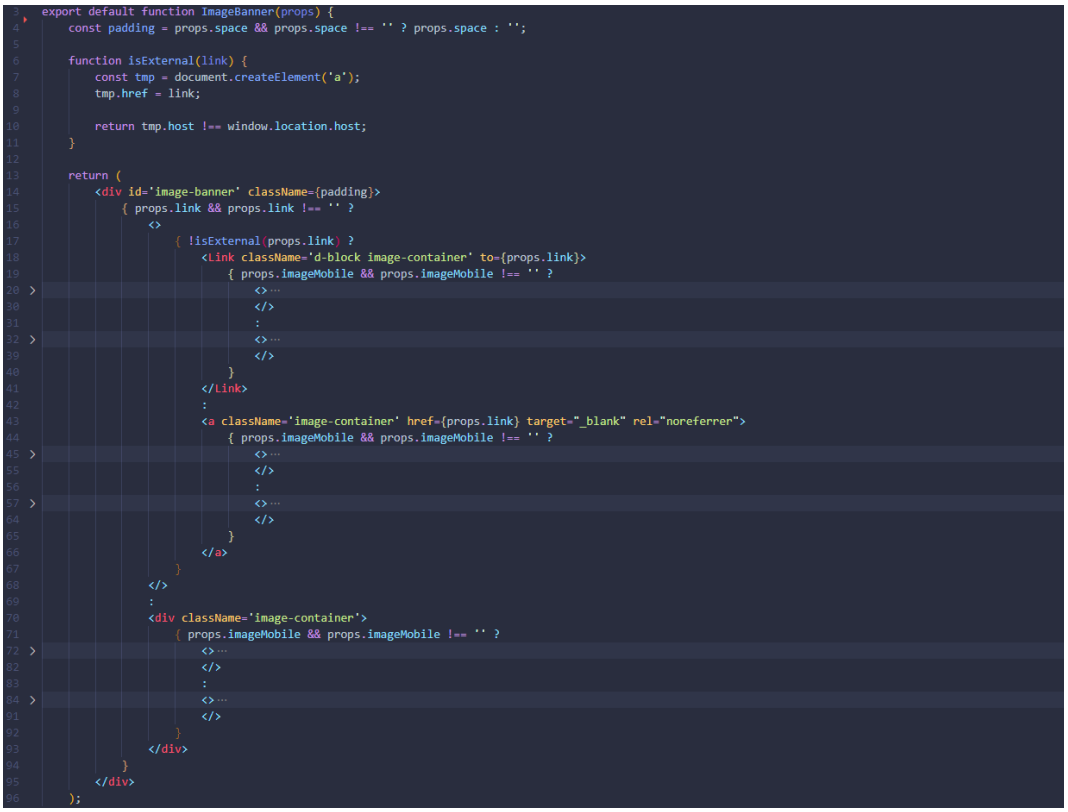
En este proyecto se ha desarrollado un componente para usar un banner personalizable, dentro de cualquiera de las vistas de la aplicación.

Este banner tiene los siguientes atributos de personalización:

- **space**: clases CSS para configurar los espacios del banner.
- **link**: link al que redirigirá el banner al ser clickado, el componente detecta de forma automática si es un link interno o externo.
- **image**: imagen de fondo que tendrá el banner, en caso de que el atributo **imageMobile** exista, cambiará a la imagen imageMobile cuando el ancho de la pantalla esté por debajo de los 768px.
- **imageMobile**: imagen de fondo para cuando el ancho de la pantalla sea menor que 768 píxeles, se puede omitir o ser vacío.
- **alt**: texto alternativo para la imagen del banner.

```
<ImageBanner
  space="pt-4 pb-5"
  link="/series/4"
  image="/assets/banner/home-better-call-saul.jpg"
  alt="Better call saul"
></ImageBanner>
```

37. Declaración del componente ImageBanner



38. Código del componente ImageBanner

4.6.6. Filtros para ordenar la información

Para **facilitar la búsqueda** de películas, series, libros, actores y autores en la aplicación, se ha desarrollado un sistema de filtros que **pueden ordenar una lista de elementos**, de forma **descendente o ascendente**, según **diferentes atributos**, por ejemplo, nombre, id, fecha…

```
handleFilter = (type, order) => {
  var filteredItems = [];

  if(type === "score"){
    filteredItems = this.state.items.sort((a, b) =>
      order === "desc" ?
        b[0] - a[0]
        :
        a[0] - b[0]
    );
  }
  else if(type === "name"){
    filteredItems = this.state.items.sort((a, b) =>
      order === "desc" ?
      b[1]["titulo"].localeCompare(a[1]["titulo"])
      :
      a[1]["titulo"].localeCompare(b[1]["titulo"])
    );
  }
  else if(type === "date"){
    filteredItems = this.state.items.sort((a, b) =>
    order === "desc" ?
        new Date(b[1]["fecha"].split('-')) - new Date(a[1]["fecha"].split('-'))
        :
        new Date(a[1]["fecha"].split('-')) - new Date(b[1]["fecha"].split('-'))
    );
  }
  else{
    filteredItems = this.state.items.sort((a, b) =>
    order === "desc" ?
        b[1][this.state.idKey] - a[1][this.state.idKey]
        :
        a[1][this.state.idKey] - b[1][this.state.idKey]
    );
  }

  this.setState({
    items: filteredItems
  })
}
```

39. Función principal para los filtros de ordenación

Por un lado, hay dos funciones, las cuales se usan para **cambiar el tipo y el orden del filtro.**

```
26
27    changeType = (event) => {
28      this.setState({ filterType: event.target.value })
29
30      this.handleFilter(event.target.value, this.state.order);
31    }
32
33    changeOrder = (event) => {
34      this.setState({ order: event.target.value })
35
36      this.handleFilter(this.state.filterType, event.target.value);
37    }
38
```

40. Funciones para los eventos de cambio en los filtros

La implementación de los filtros dentro del código HTML es la siguiente:

```
156     <div className="option-menu my-2 p-4">
157       <div className="row">
158         <div className="col-12 col-sm-6 pb-4 pb-0">
159           <div className="filter-title">Ordenar por</div>
160           <div className="d-flex align-items-center">
161             <input
162               type="radio" id="id" name="option-filter" defaultChecked value="id"
163               onClick={(e) => this.changeType(e)}
164             />
165             <label>ID</label>
166           </div>
167           <div className="d-flex align-items-center">
168             <input
169               type="radio" id="name" name="option-filter" value="name"
170               onClick={(e) => this.changeType(e)}
171             />
172             <label>Nombre</label>
173           </div>
174           <div className="d-flex align-items-center">
175             <input
176               type="radio" id="fecha" name="option-filter" value="date"
177               onClick={(e) => this.changeType(e)}
178             />
179             <label>Fecha</label>
180           </div>
181         </div>
182         <div className="col-12 col-sm-6">
183           <div className="filter-title">Ordenar de</div>
184           <div className="d-flex align-items-center">
185             <input
186               type="radio" id="desc" name="option-order" value="desc"
187               onClick={(e) => this.changeOrder(e)}
188             />
189             <label>Mayor a menor</label>
190           </div>
191           <div className="d-flex align-items-center">
192             <input
193               type="radio" id="asc" name="option-order" defaultChecked value="asc"
194               onClick={(e) => this.changeOrder(e)}
195             />
196             <label>Menor a mayor</label>
197           </div>
198         </div>
199       </div>
200     </div>
```

41. Código de implementación de los filtros

4.6.7. Clases CSS para configurar el aspect-ratio de una imagen

Una de las funcionalidades más útiles de este proyecto, ha sido crear diferentes clases CSS, las cuales, al ser **aplicadas a imágenes**, hacen que estas **conserven**, en todo momento, la **proporcionalidad o aspecto-ratio,** independientemente del ancho de la pantalla o contendor.

Para elegir una aspecto-ratio, se debe poner la etiqueta de la imagen, dentro de un contenedor con la clase “ratio-container” más el aspecto-ratio que queramos, por ejemplo, “ratio16-9”, si, además, añadimos la clase “fill-img” a este contenedor, la imagen se ajustará a todo el ancho y largo de este.

```
.ratio-container{
    display: block;
    position: relative;

    &.ratio1-1{
        padding-bottom: calc(1 / 1 * 100%);
    }
    &.ratio3-1{
        padding-bottom: calc(1 / 3 * 100%);
    }
    &.ratio2-3{ ···
    }
    &.ratio3-2{ ···
    }
    &.ratio3-4{ ···
    }
    &.ratio4-3{ ···
    }
    &.ratio16-9{ ···
    }
    &.ratio21-8{ ···
    }
    &.ratio9-16{ ···
    }
    &.ratio10-16{ ···
    }

    img{
        position: absolute;

        object-fit: cover;
        height: 100%;
        width: 100%;
    }

    &.fill-img{
        img{ object-fit: fill; }
    }
}
```

42. Clases CSS para el aspect-ratio de una imagen

```
<div className='ratio-container ratio9-16 fill-img'>
    { props.photo && props.photo !== "" ?
        <img src={"/assets/"+ props.photo} alt={props.title}/>
        :
        <img src="/assets/image-placeholder.jpg" alt={props.title}/>
    }
</div>
```

43. Uso de las clases CSS para el aspect-ratio de una imagen

4.7. Problemas encontrados y solución

Durante el desarrollo del proyecto, se han **encontrado** algunos **problemas**. A continuación, se **mostrarán y se explicará la solución** que se ha aplicado para resolverlos.

4.7.1. Rutas absolutas en ReactJS

Al principio del proyecto, la **importación de archivos**, **componentes**, **imágenes**, etc, llegaba a ser demasiado complicada, ya que, escribir la ruta al recurso que se quería importar, daba lugar, en muchas ocasiones a **errores o equivocaciones.**

Por ello, se ha configurado la aplicación de ReactJS, para que, a la hora de **importar un nuevo recurso**, pueda hacerse **mediante rutas absolutas** de las carpetas raíz del proyecto, dentro de la carpeta donde se encuentra la lógica (/src).

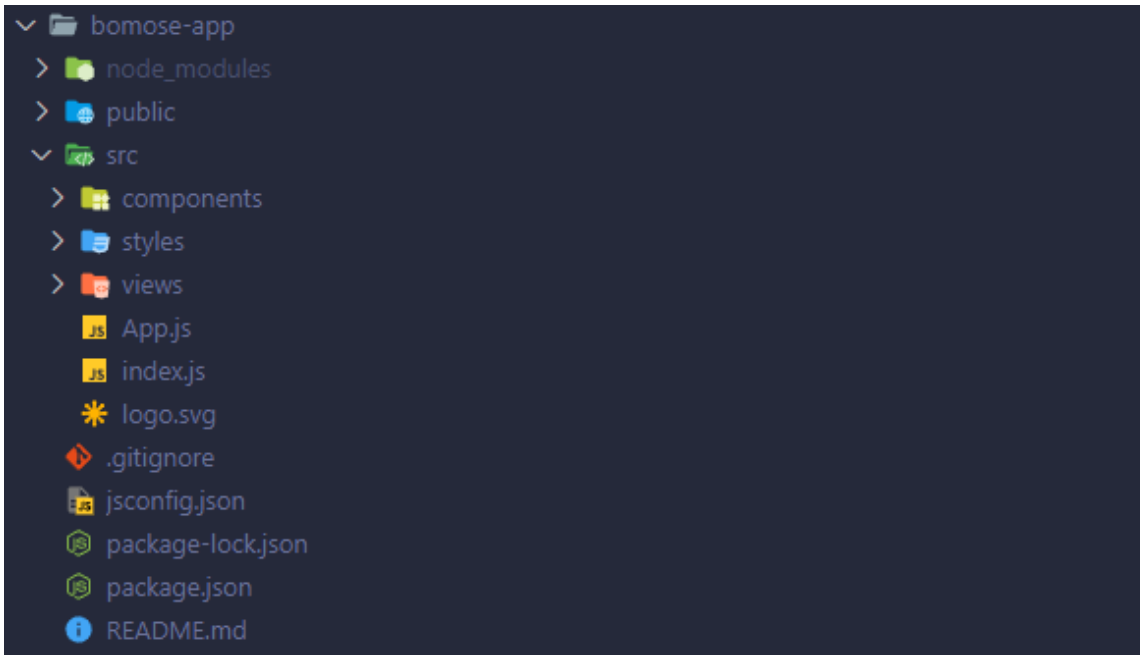
Para conseguir esto, se ha añadido el archivo jsconfig.json dentro de la aplicación, con el siguiente código:

```
{
    "compilerOptions": {
      "baseUrl": "src"
    },
    "include": ["src"]
}
```

44. Configuración de rutas absolutas en ReactJS

Estas carpetas raíz son las siguientes:

- **components:** archivos con la lógica de los componentes de la aplicación.
- **styles:** archivos SCSS, con los estilos para las vistas y componentes de la aplicación.
- **views:** archivos con la lógica para las vistas de la aplicación.



45. Estructura de carpetas en la aplicación de ReactJS

A continuación, se muestra un ejemplo de cómo, tras haber realizado lo anteriormente escrito, se realizan las importaciones de recursos dentro de la aplicación.

```
import React from "react";
import { Routes, Route, Navigate, useNavigate } from "react-router-dom";

//COMPONENTS
import { Header } from "components/Header";
import Footer from "components/Footer";
import Head from "components/Head";

//VIEWS
import Home from "views/Home";
import { Login } from "views/Login";
import { Signin } from "views/Signin";
import { Profile } from 'views/Profile';
import Item from "views/Item";
import UpdateItem from "views/UpdateItem";
import { CreateItem } from "views/CreateItem";
import UpdateArtist from "views/UpdateArtist";
import { CreateArtist } from "views/CreateArtist";
import { HomeArtists } from "views/HomeArtists";
import User from "views/User";
import Artist from "views/Artist";
import UpdateUser from "views/UpdateUser";
import { HomeItems } from 'views/HomeItems';
import Error404 from 'views/Error404';

//STYLESHEETS
import "styles/globals.scss";
import "styles/variables.scss";
import 'bootstrap-icons/font/bootstrap-icons.css';
```

46. Uso de rutas absolutas en ReactJS

4.7.2. Error al actualizar la información en tablas muchos a muchos

Realizando diversas pruebas en la aplicación, se detectó, que cuando se realizaba la actualización de una tabla, que tuviera una relación muchos a muchos, las filas que participaban en la relación se eliminaban.

Por ejemplo, cuando se actualizaba la información de una película, serie o libro, se eliminaban las filas que participaban en la relación, en este caso, se eliminaban las filas de actores o autores.

Para solucionar este error, se ha **cambiado**, dentro de las clases que representan las tablas de relaciones muchos a muchos, uno de los **parámetros** que se usan con la **notación @ManyToOne** de Spring Boot, cascade, este parámetro, en un principio, estaba con el valor CascadeType.ALL y se cambió a CascadeType.DETACH para solucionar el error.

```
17 @Entity
18 @Table(name = "RepartoPelicula")
19 @IdClass(value = RepartoPeliculaId.class)
20 public class RepartoPelicula implements Serializable {
21
22     @Id
23     @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.DETACH)
24     @JsonIncludeProperties({"idActor", "nombre", "apellidos"})
25     @JoinColumn(name="id_actor")
26     private Actor actor;
27
28     @Id
29     @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.DETACH)
30     @JsonIncludeProperties({"idPelicula", "titulo", "foto"})
31     @JoinColumn(name="id_pelicula")
32     private Pelicula pelicula;
```

47. Configuración de la notación @ManyToOne para relaciones muchos a muchos

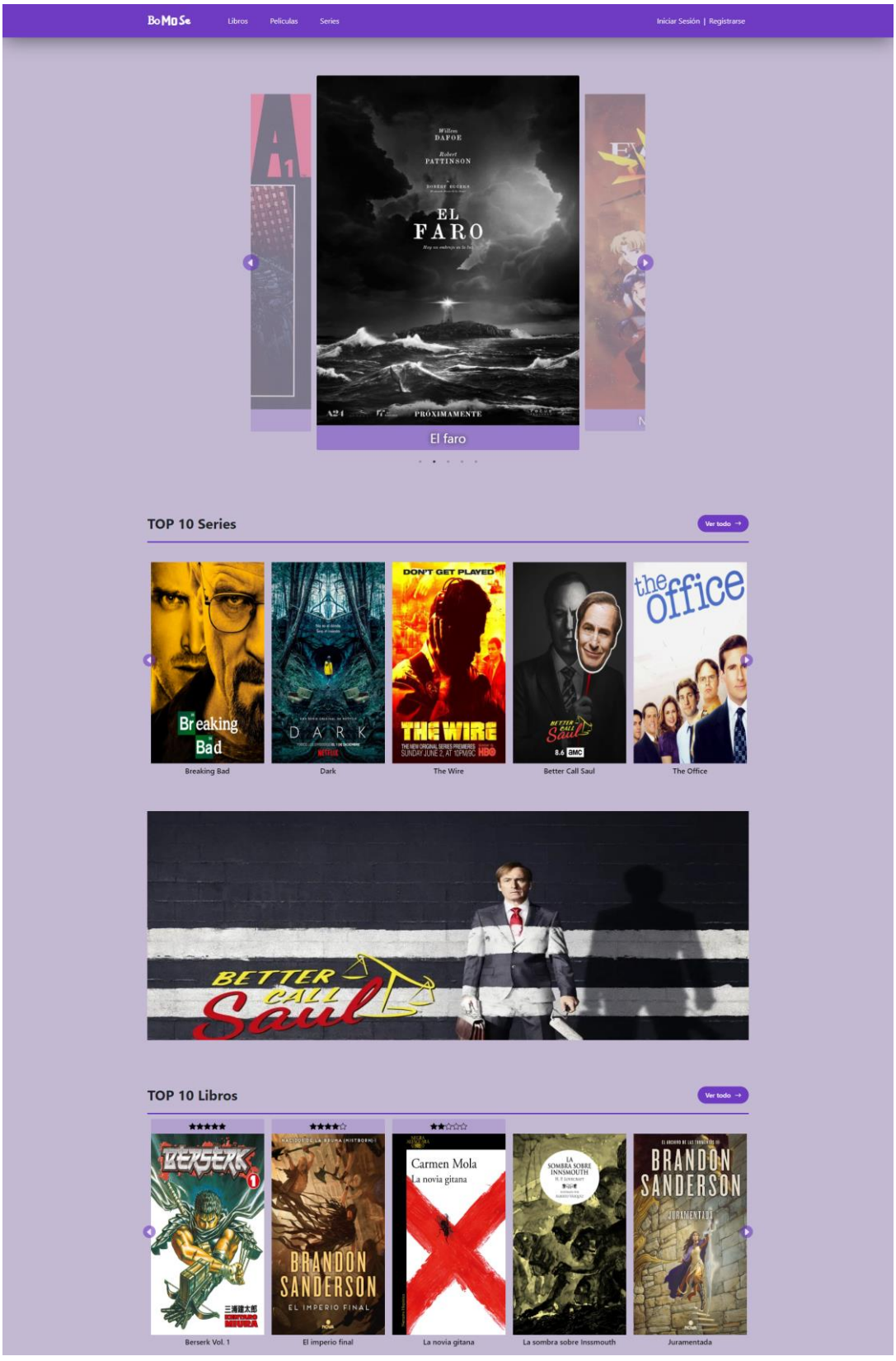
5. Manual de usuario

En este apartado se redactará un manual, para facilitar a los usuarios el uso de la página web, además de mostrar todas las funcionalidades implementadas.

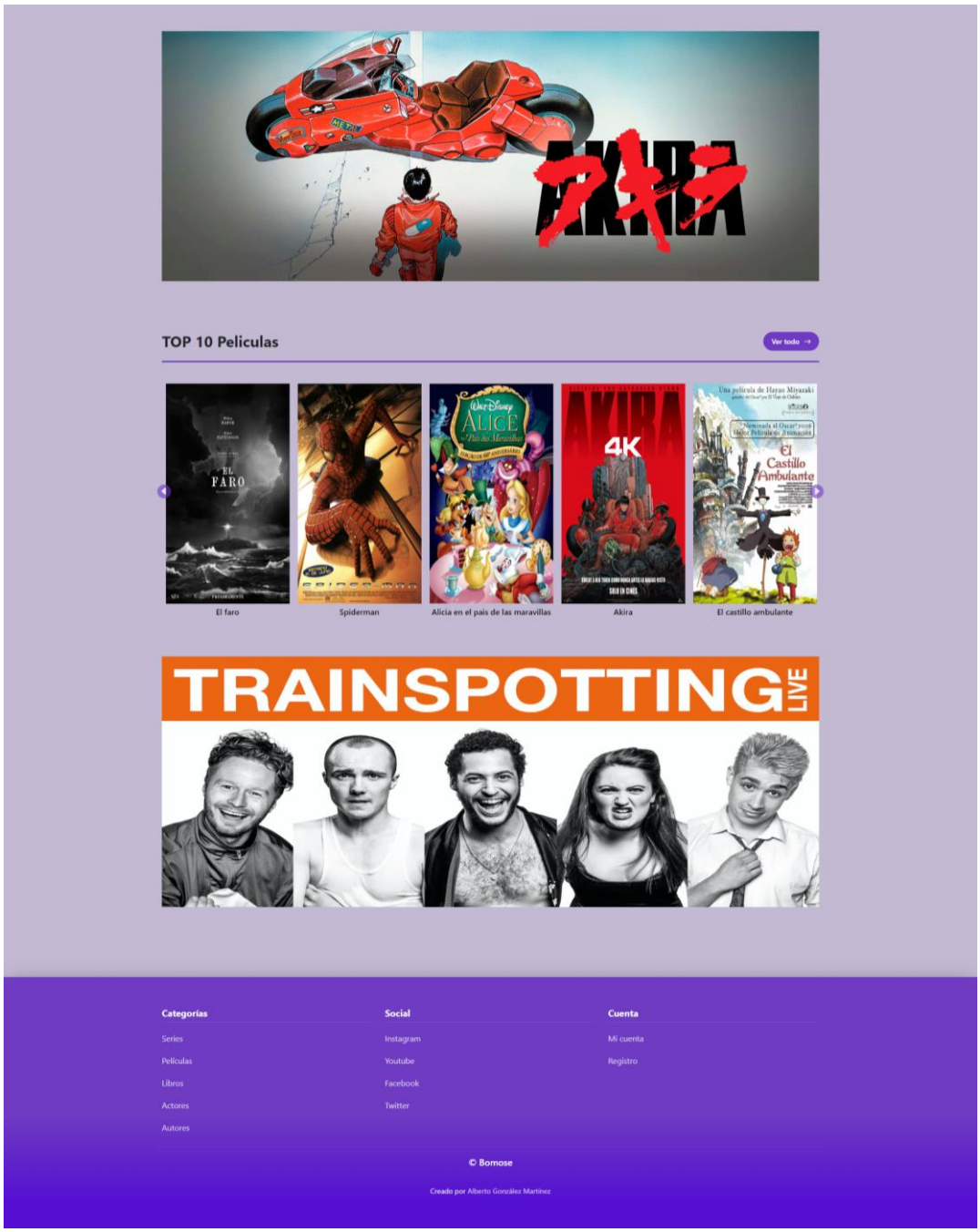
En este manual solo se mostrarán las capturas de la página web en ordenador, pero, como se ha explicado anteriormente, la página es perfectamente navegable en otros dispositivos, independientemente de sus dimensiones.

5.1. Vista principal (Home)

La **vista principal** de la aplicación es la **Home**, en esta página, hay numerosos módulos que muestran al usuario que libros, películas o series se recomiendan o están destacados, banners destacados, además del top 10 mejor valorados de la aplicación.



48. Página Home pt.1



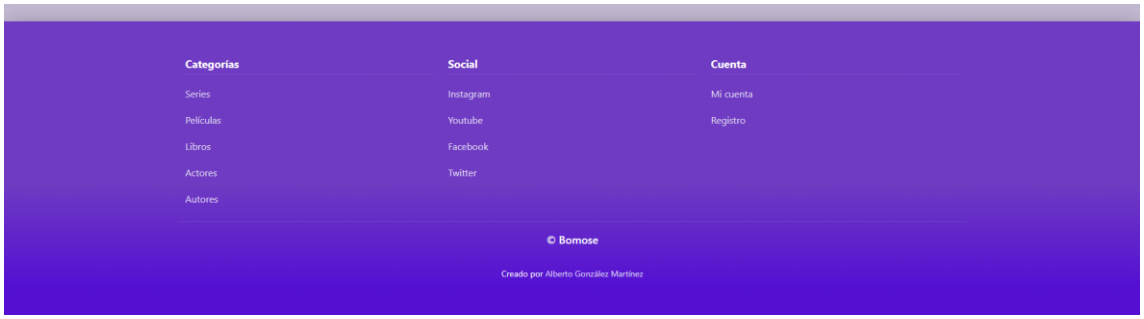
49. Página Home pt.2

Desde cualquiera de las páginas se podrá acceder, en todo momento, a las demás páginas principales. Esto se debe a que, siempre están visibles para el usuario la barra de navegación y el *footer*.

Con la **barra de navegación**, se puede navegar a la página con el listado completo de series, libros o películas, además se puede iniciar sesión o registrarse en la aplicación.

50. Barra de navegación de la página web

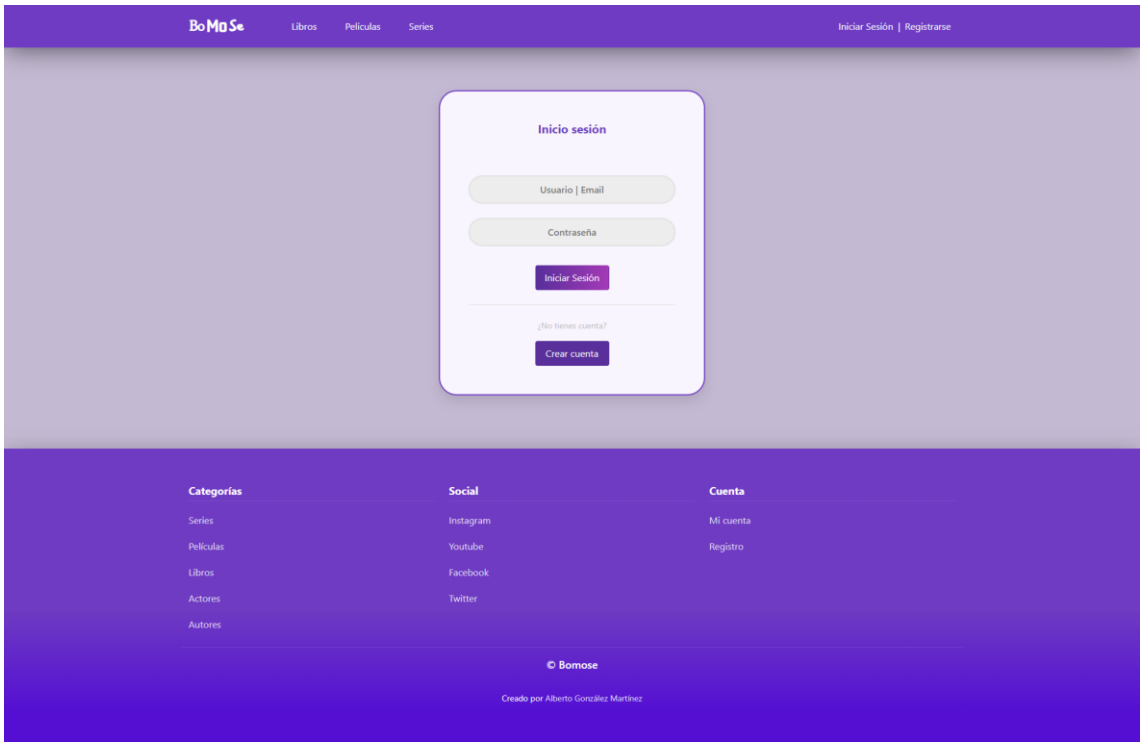
Con el ***footer***, se puede navegar a la página con el listado completo de series, libros, películas, autores o actores, además se puede acceder al registro y perfil de usuario.



51. Footer de la página web

5.2. Vista inicio de sesión

Para poder tener **acceso a todas las funcionalidades** de la aplicación, el **usuario** tiene que **iniciar sesión**. La vista de inicio de sesión consta de un formulario con dos campos, uno para introducir el nombre de usuario o email y otro para la contraseña, además de dos botones, uno para iniciar sesión y otro para acceder a la vista de crear una nueva cuenta.



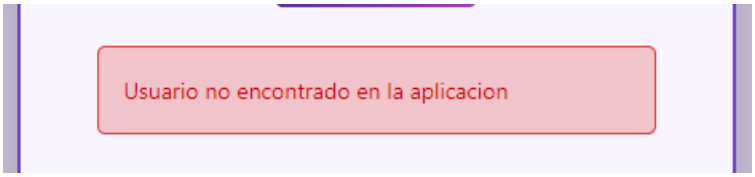
52. Página inicio de sesión

El usuario debe **introducir** el **nombre de usuario o email** y la **contraseña** de su cuenta, una vez hecho esto, debe **pulsar** en el **botón "Iniciar Sesión".**

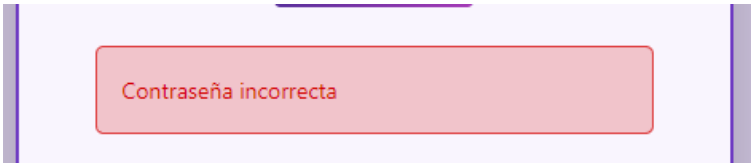
Si todo **ha salido bien**, se redirigirá al usuario a la **última página visitada** y se mostrará en la **barra de navegación** el **usuario** que ha iniciado sesión y se dará la opción de acceder al perfil o cerrar sesión.

53. Opciones de la barra de navegación para usuario registrado

En caso de que el usuario haya introducido algún campo mal saldrá uno de los siguientes **avisos de error:**



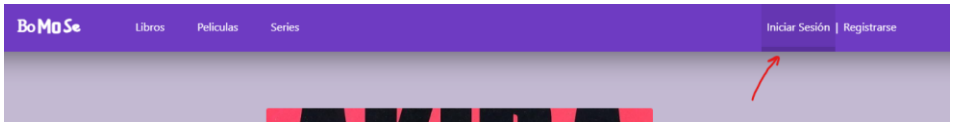
54. Error usuario no encontrado al iniciar sesión



55. Error contraseña incorrecta al iniciar sesión

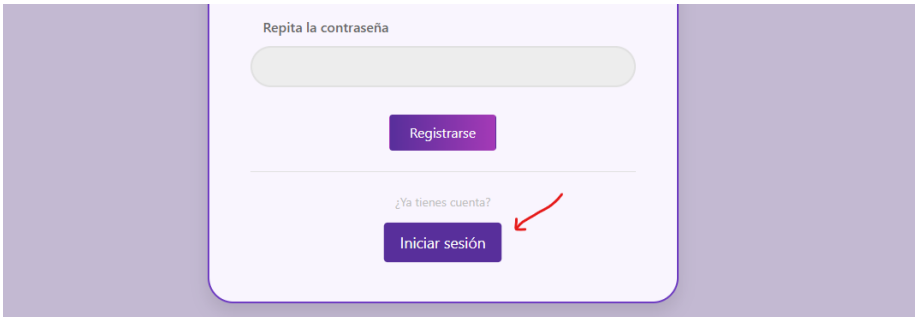
Para entrar en la página de inicio de sesión, el usuario puede acceder de distintas formas:

- Mediante la **barra de navegación**: se puede acceder haciendo click en el link “Iniciar Sesión” de la barra de navegación.



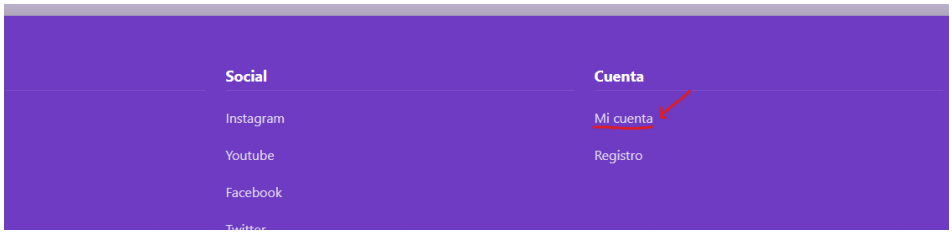
56. Link iniciar sesión de la barra de navegación

- En la **vista de registro**: si el usuario accede a la vista de registro, pero, ya tiene una cuenta, puede acceder al inicio de sesión haciendo click en el botón de “Iniciar sesión”.



57. Botón iniciar sesión de la página de registro

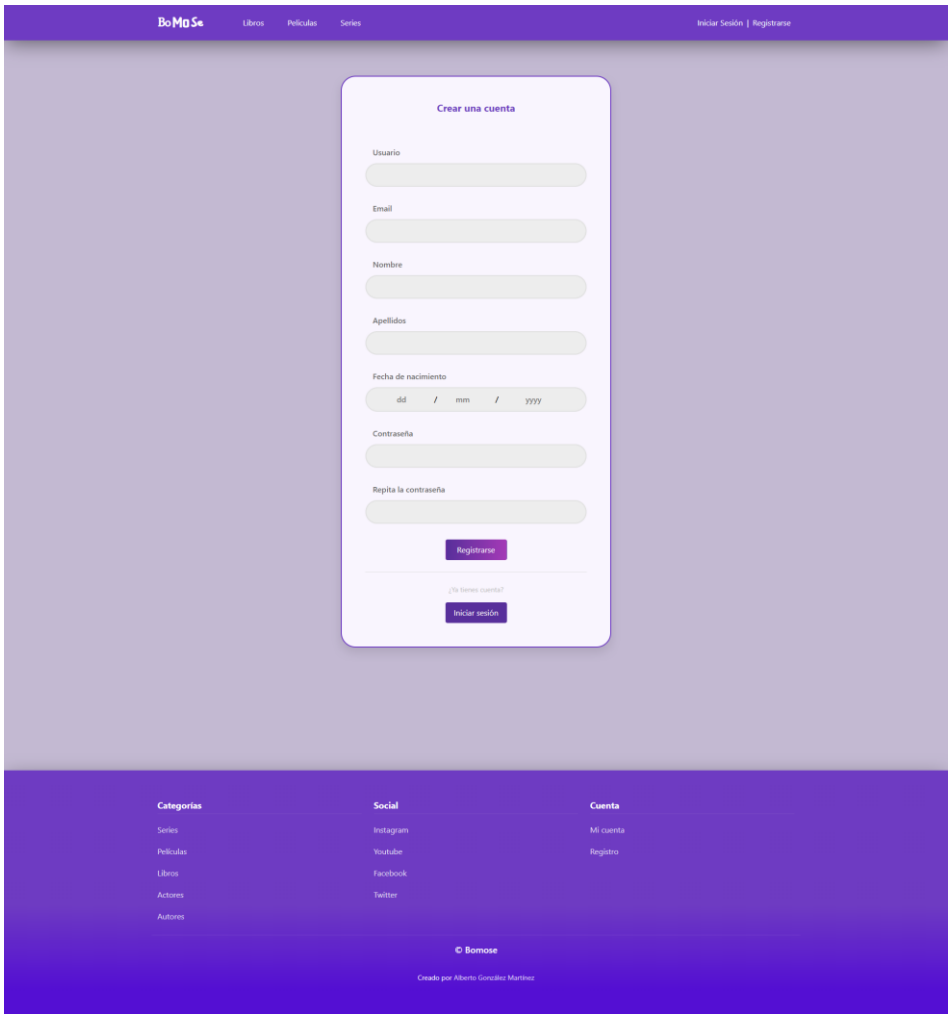
- Mediante el ***footer***: si el usuario hace click en el link “Mi cuenta” y no ha iniciado sesión, se le redirigirá al inicio de sesión.



58. Link mi cuenta del pie de página

5.3. Vista registro

Si el **usuario no tiene una cuenta creada** y quiere acceder a todas las funcionalidades, tiene que **registrarse en la aplicación**.



59. Página registro de usuario

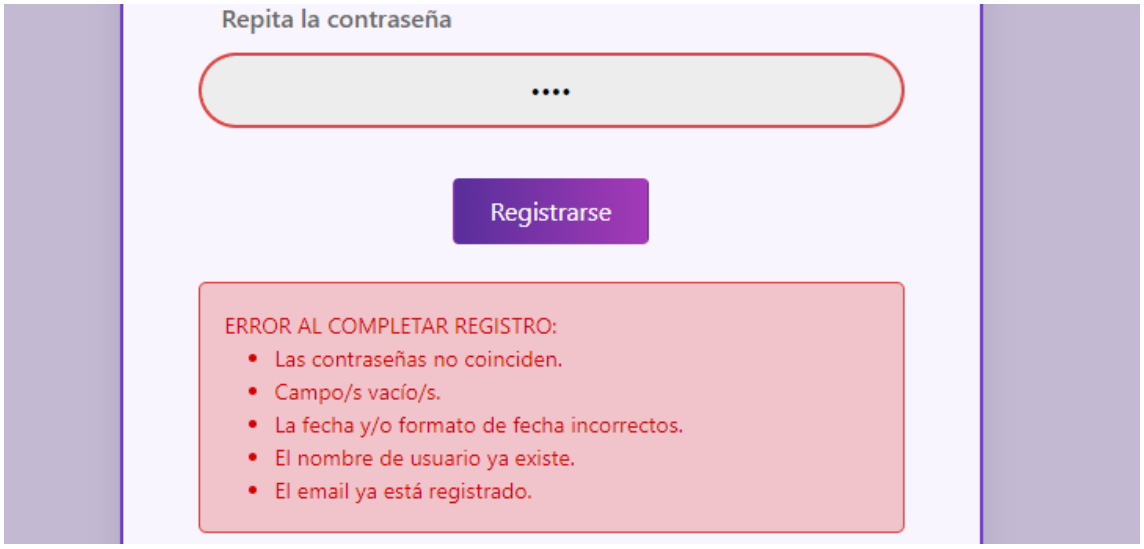
La vista para crear una cuenta consta de un formulario con todos los campos, los cuales, deben rellenarse obligatoriamente, y dos botones, uno para completar el registro y otro para acceder a la vista de inicio de sesión.

El usuario debe **introducir todos los datos del formulario** y, una vez hecho esto, **pulsar el botón "Registrarse".**

Si todo **ha salido bien** el usuario será **redirigido a la home** y, en la **barra de navegación**, saldrá el **nombre** del nuevo **usuario creado**, con la posibilidad de acceder al perfil o cerrar sesión.

60. Opciones de la barra de navegación para nuevo usuario

En caso de que el usuario haya **introducido algún campo mal** saldrá un **aviso** de **error personalizado**, con los errores que el usuario tiene que corregir y, además, se señalarán que campos son los que contienen algún error:



61. Mensaje de error personalizado para registro de usuario

A parte de la validación de los campos del formulario, se ha implementado una comprobación **en tiempo real** para que el usuario sepa si el **nombre de usuario** o el **email** que está introduciendo, **ya están registrados**. Los avisos se muestran de la siguiente forma:

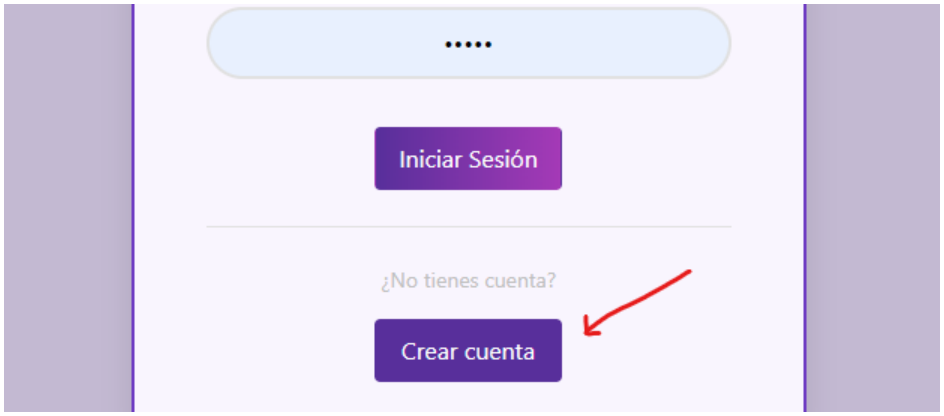
62. Error nombre de usuario y/o email ya registrados

Para entrar en la página de inicio de sesión, el usuario puede acceder de distintas formas:

- Mediante la **barra de navegación**: se puede acceder haciendo click en el link “Registrarse” de la barra de navegación.

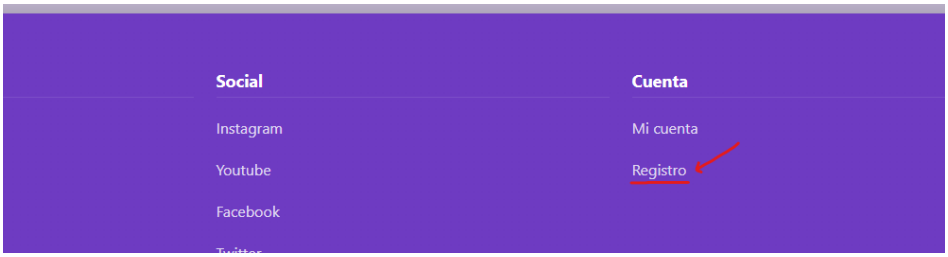
63. Link registrarse de la barra de navegación

- En la vista de **inicio de sesión**: si el usuario accede a la vista de inicio de sesión, pero, no tiene cuenta y quiere crear una, puede acceder al registro haciendo click en el botón de “Crear cuenta”.



64. Botón crear cuenta de la página inicio de sesión

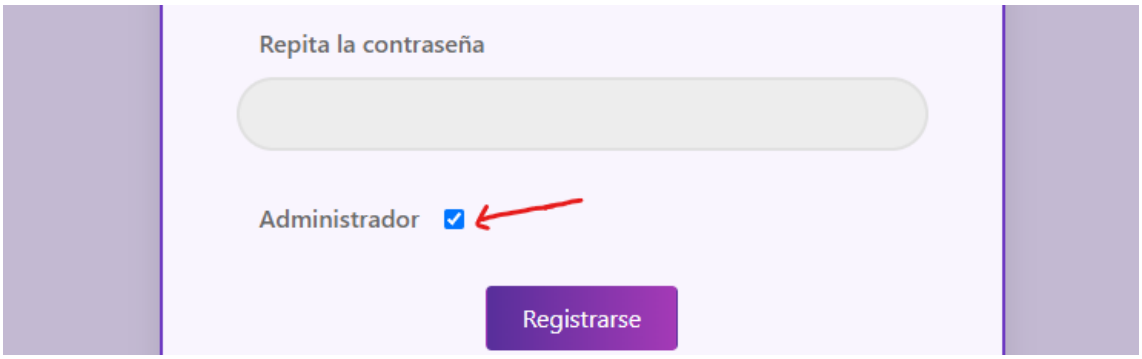
- Mediante el **footer**: si el usuario hace click en el link “Registro”, se le redirigirá al registro de usuario.



65. Link registro del pie de página

5.3.1. Funcionalidades de administrador

Si el **usuario** que accede a esta vista **es un administrador** de la aplicación, también, saldrá la opción de **hacer administrador al nuevo usuario** creado, para ello, el administrador tiene que marcar la casilla de “Administrador”:

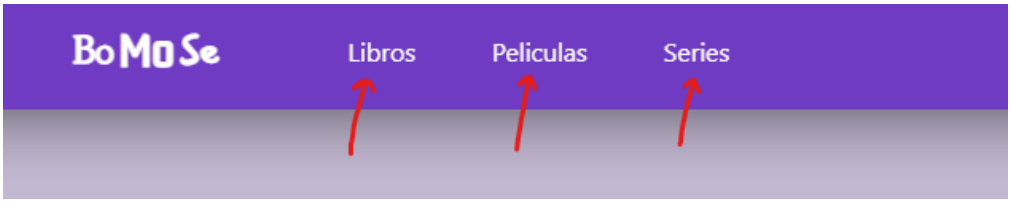


66. Checkbox administrador de la página de registro

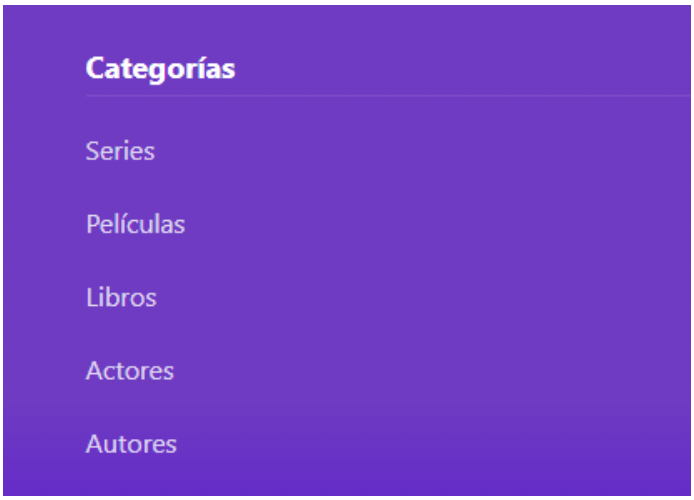
5.4. Vista principal de libros, películas, series, actores y autores

La **vista** principal para **series, películas, libros, actores y autores** consta de una lista con la **información de cada elemento,** además, esta lista puede ser ordenada mediante filtros.

Para **acceder a esta vista** hay que hacer click en los links de la barra de navegación, o desde los links de la parte de “Categorías” del footer.

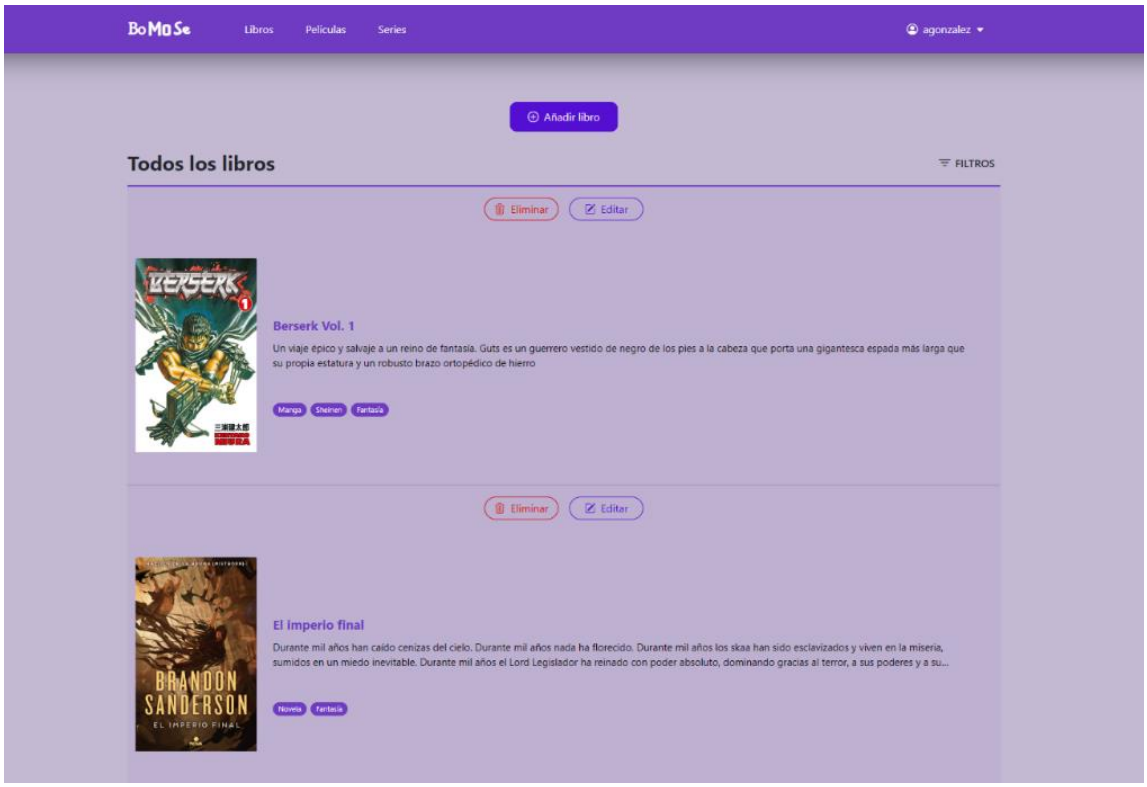


67. Links para libros, películas y series de la barra de navegación



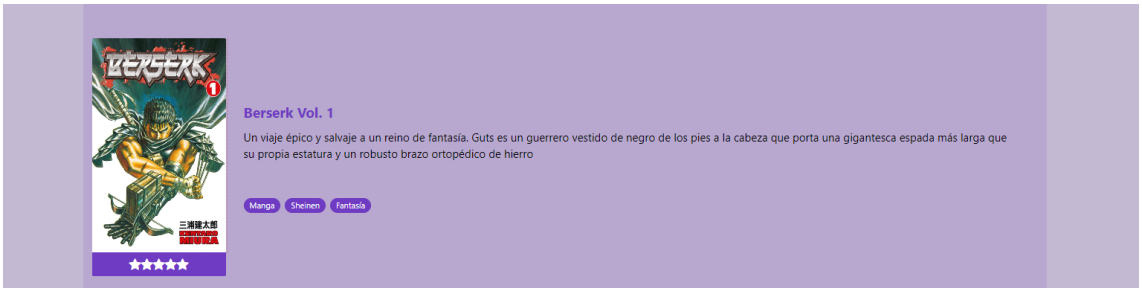
68. Links de categorías del pie de página

La **información** que se muestra de cada elemento, en el caso de **series, películas y libros** es, una foto de la portada, el título, la descripción y un conjunto de etiquetas.



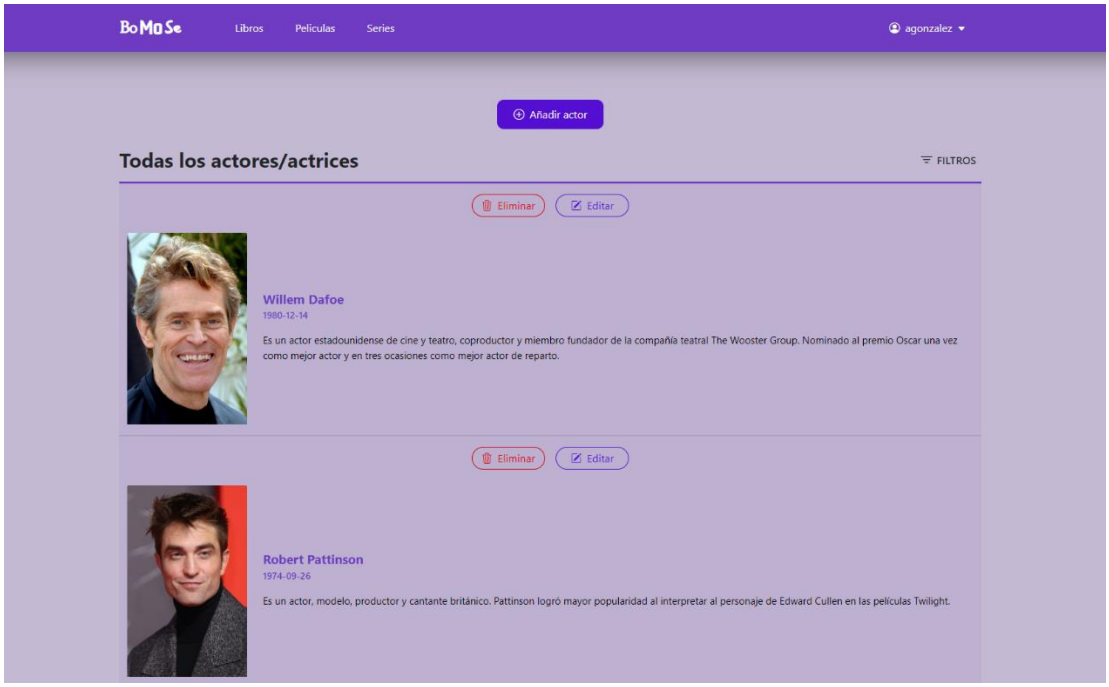
69. Página principal para libros

Cuando se pasa el **ratón por encima** de alguno de los elementos, se muestra cual es la **nota media** de este.



70. Elemento de la lista de libros

En el caso de la **información** que se muestra sobre **actores y autores**, es la foto del actor o autor, el nombre, el año de nacimiento y una descripción.



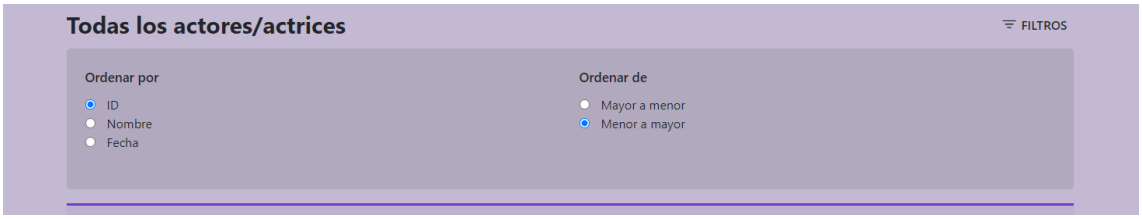
71. Página principal para actores

Al **hacer click** en cualquiera de los elementos de la lista, el usuario será **redirigido** a la **vista** que **muestra la información completa** del elemento.

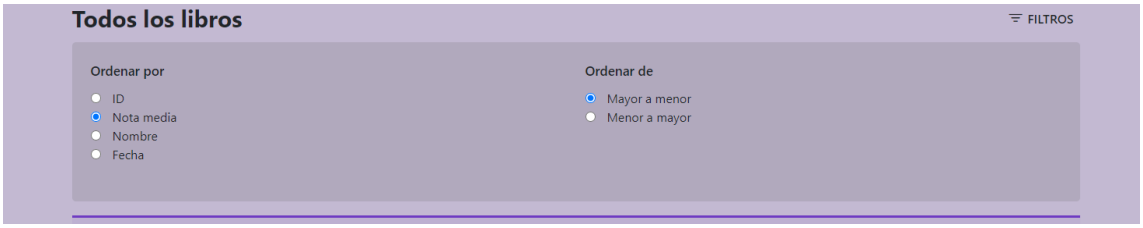
Para **facilitar la búsqueda** entre los elementos de la lista, el usuario puede seleccionar diferentes **filtros de ordenación**. Primero, se debe pulsar en el botón “FILTROS” que se encuentra en la parte superior derecha de la lista.

72. Botón para desplegar los filtros

Una vez pulsado el botón se desplegarán las opciones de los filtros de ordenación, al hacer click en cualquiera de las opciones, la lista se ordenará de forma automática.



73. Opciones de los filtros de actores y actrices

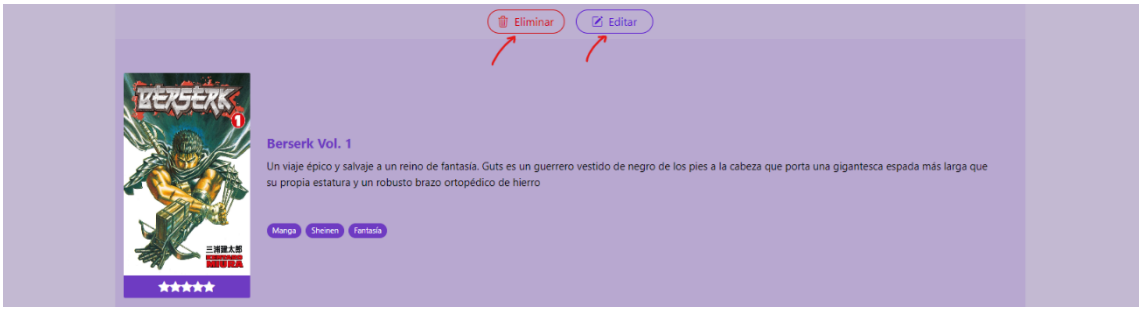


74. Opciones de los filtros de libros, series y películas

5.4.1. Funcionalidades de administrador

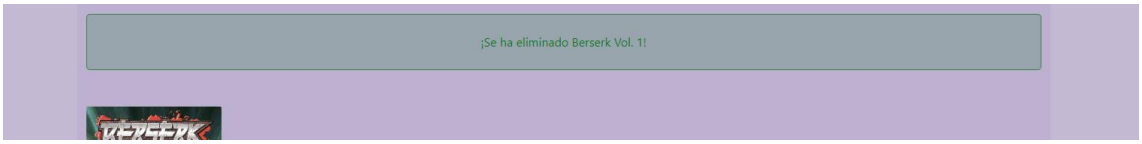
Si el **usuario** que accede a esta vista es un **administrador** de la aplicación, tendrá disponibles las siguientes funcionalidades.

El administrador podrá **editar la información de un elemento** al hacer click en el botón “Editar”, este botón redirigirá al administrador a la vista para editar el elemento.



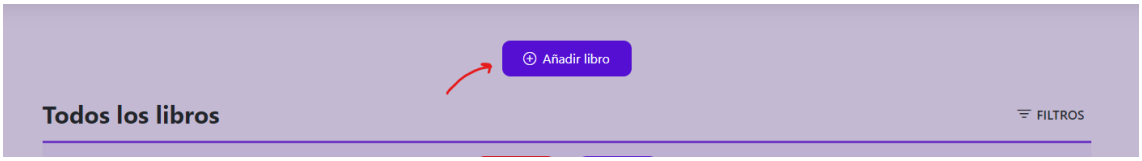
75. Botones para eliminar y editar un elemento en la página principal de libros

También podrá **borrar un elemento** directamente desde la lista al hacer click en el botón “Eliminar”. Si todo ha salido bien, saldrá un mensaje avisando al administrador que se ha eliminado el elemento.



76. Mensaje de confirmación para elemento eliminado

Por último, se podrá **añadir un nuevo elemento**, para ello se debe hacer click en el botón “Añadir”, este botón redirigirá al administrador a la vista para añadir un nuevo elemento.



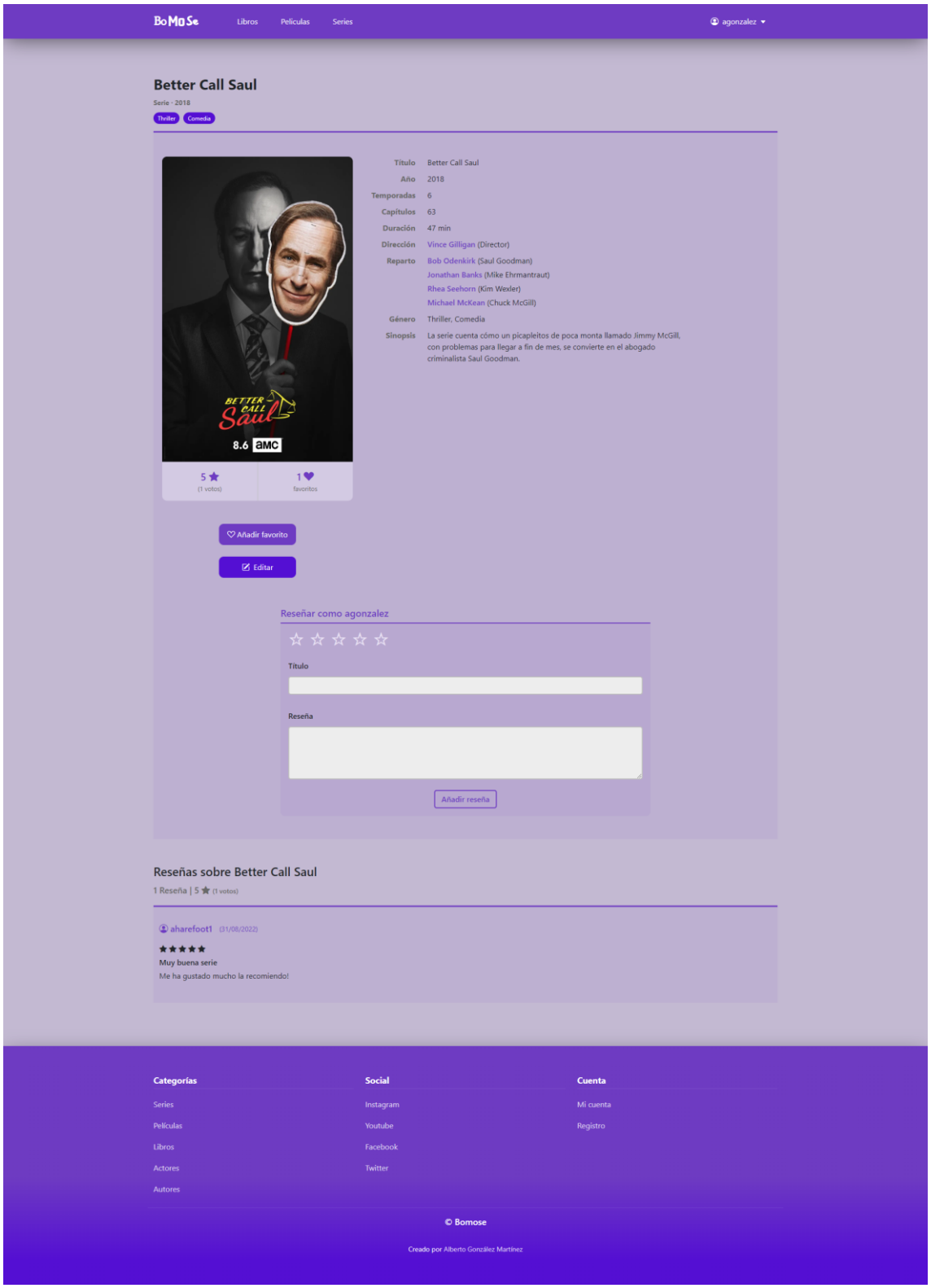
77. Botón añadir elemento de la página principal de libros

5.5. Vista información de libros, películas, series

La vista para **consultar** una de las **películas, series o libros** contiene la información y diferentes funcionalidades, las cuales se explicarán más adelante.

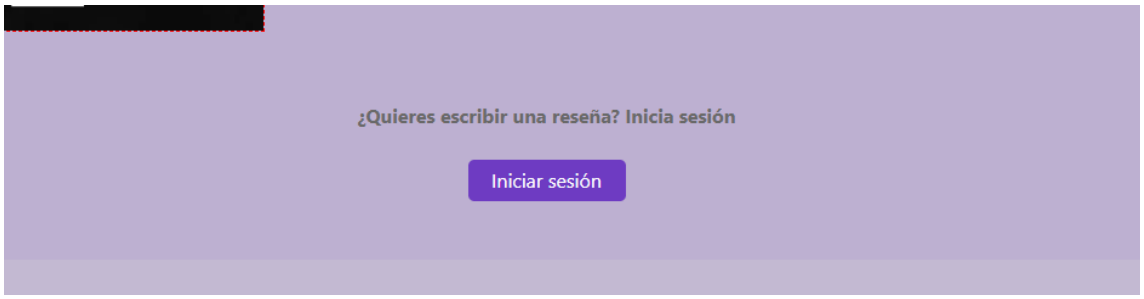
En el **panel principal** contiene, la imagen de portada junto con la nota media, el número de votos y las veces que ha sido marcado como favorito, también la información completa sobre la obra, junto a la lista con el reparto y dirección de esta.

La lista con el **reparto y dirección o autoría** de la obra está compuesta por links con el nombre del actor o autor y el papel o rol que tiene dentro de la obra. Al **hacer click** en estos links, el **usuario** será **redirigido** a la **vista** con la **información sobre el autor o actor.**



78. Página para la información de libros, series y películas

Si el **usuario no está registrado** dentro de la aplicación, únicamente podrá ver la información y la lista de reseñas, además saldrá este mensaje advirtiendo al usuario de que para realizar una reseña tendrá que iniciar sesión.



79. Mensaje de advertencia sobre escribir una reseña a usuario no registrado

Al final de esta vista, se puede ver la **lista** con todas las **reseñas realizadas por los usuarios de la aplicación**, en esta lista, solo estarán las reseñas que tengan, al menos, título o descripción, en caso de que solo tenga una nota, contará únicamente para hacer la nota media de la obra.

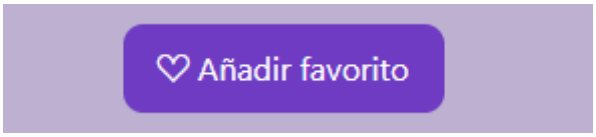
Cada una de las reseñas de la lista tiene un link con el nombre del usuario que ha realizado la reseña, la fecha cuando se ha realizado la reseña, la nota y texto de esta. Si el usuario hace click en el nombre de usuario, se le redirigirá a la vista con la información sobre el usuario.

80. Lista de reseñas

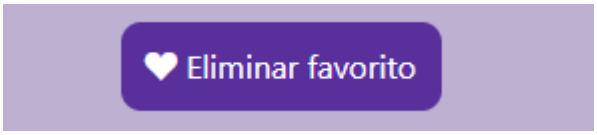
5.5.1. Funcionalidades usuario registrado

Si el **usuario ha iniciado sesión,** tendrá la opción de **marcar** un elemento como **favorito** y **hacer una reseña** de este.

Para **añadir** el elemento **como favorito**, el usuario debe hacer click en el botón "Añadir favorito", una vez clickado, cambiará su estado y el usuario podrá eliminar el elemento como favorito volviendo hacer click en él.



81. Botón añadir favorito

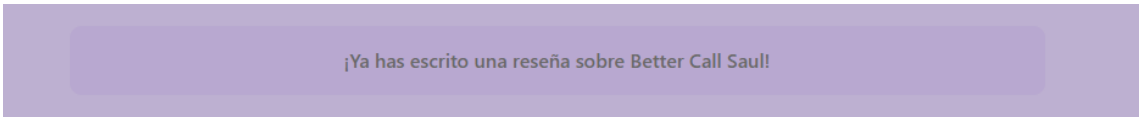


82. Botón eliminar favorito

Para **hacer una reseña** el usuario deberá, al menos, introducir una nota, de uno a cinco estrellas y, posteriormente, hacer click en el botón "Añadir reseña".

83. Formulario para realizar una reseña

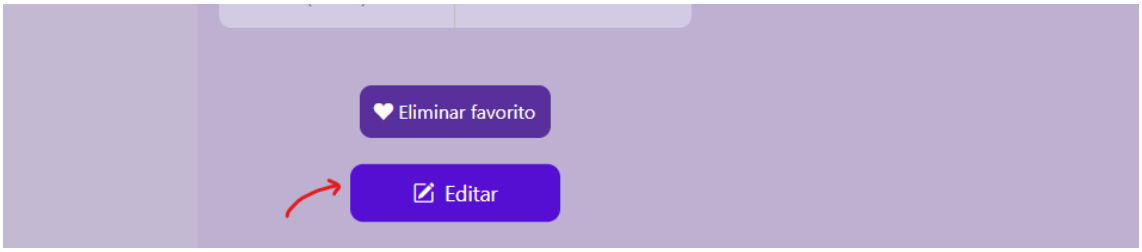
Una vez que el usuario ha realizado una reseña sobre la obra, ya no tendrá la opción de volver a realizar una reseña y aparecerá este mensaje en la vista:



84. Mensaje de reseña ya realizada

5.5.2. Funcionalidades de administrador

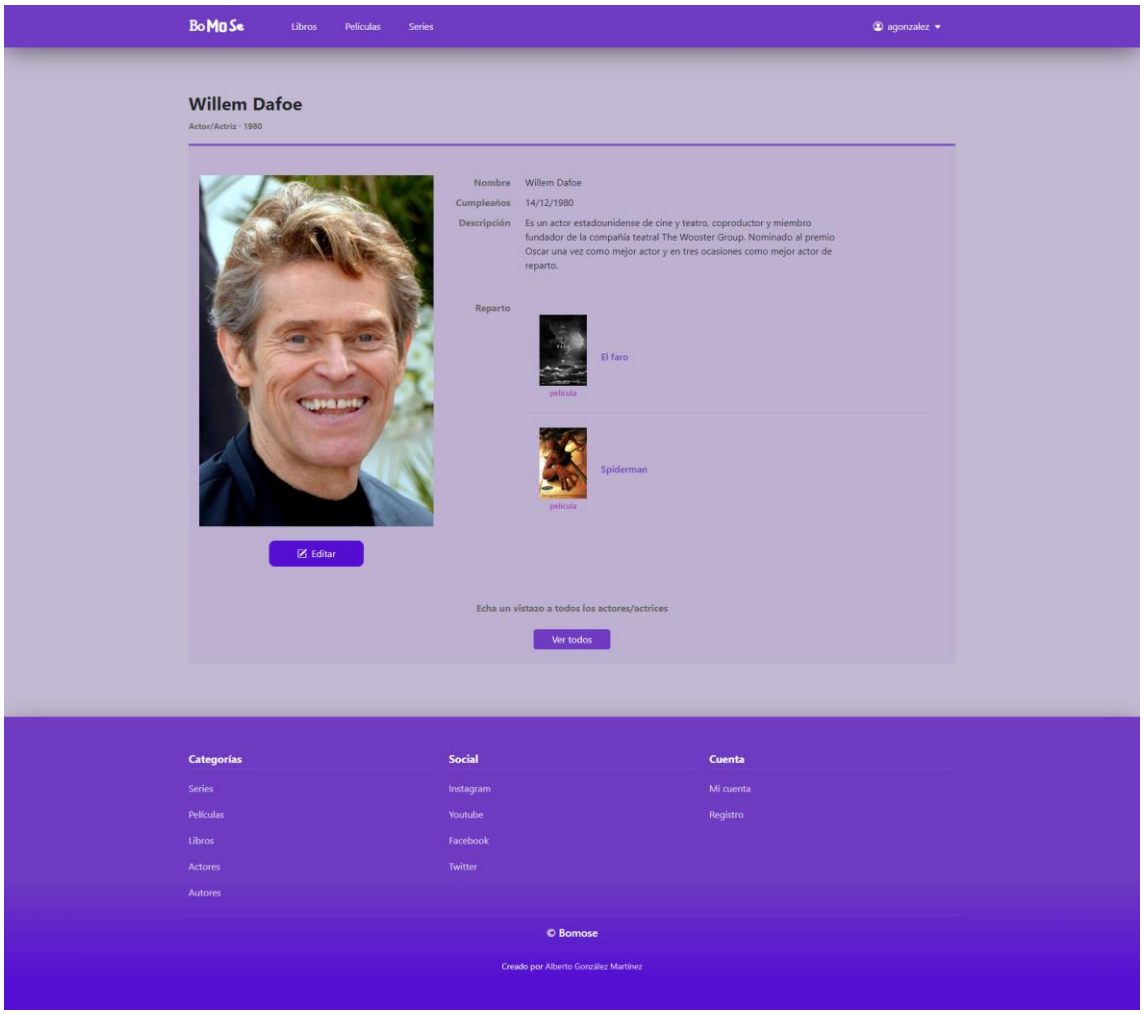
Si el **usuario** que accede a la aplicación **es un administrador**, tendrá la opción de editar la información de la obra. Para ello, se debe hacer click en el botón "Editar", el cual redirigirá al usuario a la vista para editar la información sobre películas, series o libros.



85. Botón editar información de película, libro o serie

5.6. Vista información actores y autores

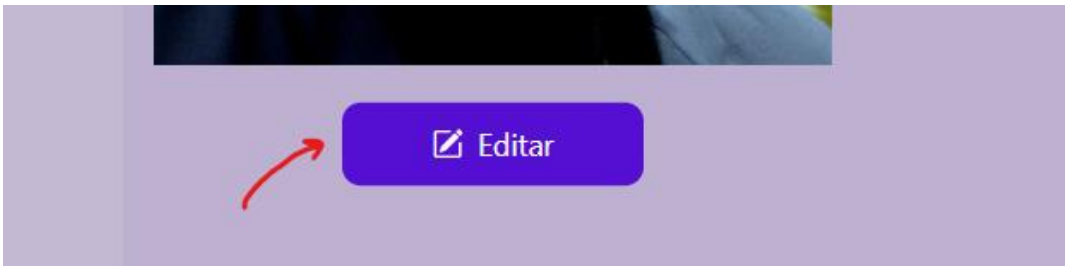
La vista para **consultar la información sobre autores o actores** contiene un panel con la foto del actor o autor, la información completa y una lista con las obras en las que ha participado como director o actor. Los elementos de esta lista pueden ser clickados y redirigirán al usuario a la vista con la información de esa obra.



86. Página para la información de actores y autores

5.6.1. Funcionalidades de administrador

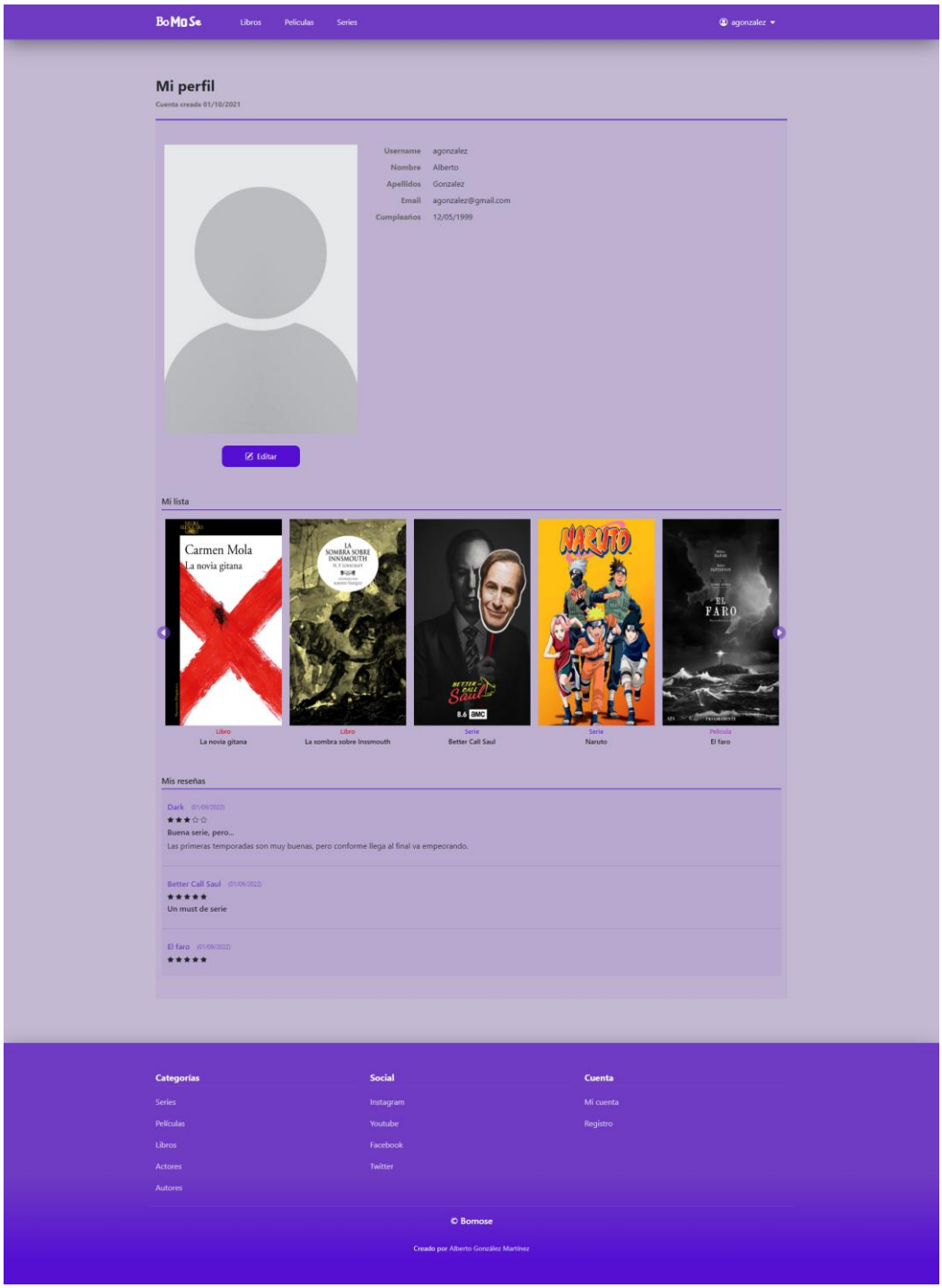
Si el **usuario** que accede a la aplicación **es un administrador**, tendrá la opción de **editar la información del artista**. Para ello, se debe hacer click en el botón "Editar", el cual redirigirá al usuario a la vista para editar la información sobre actores o autores.



87. Botón editar información de actores y autores

5.7. Vista mi perfil

Esta vista solo es **accesible** cuando el **usuario de la aplicación ha iniciado sesión**, en esta vista, se puede consultar la información referente al usuario, el listado de obras marcadas como favorito y el listado de reseñas realizadas por este.



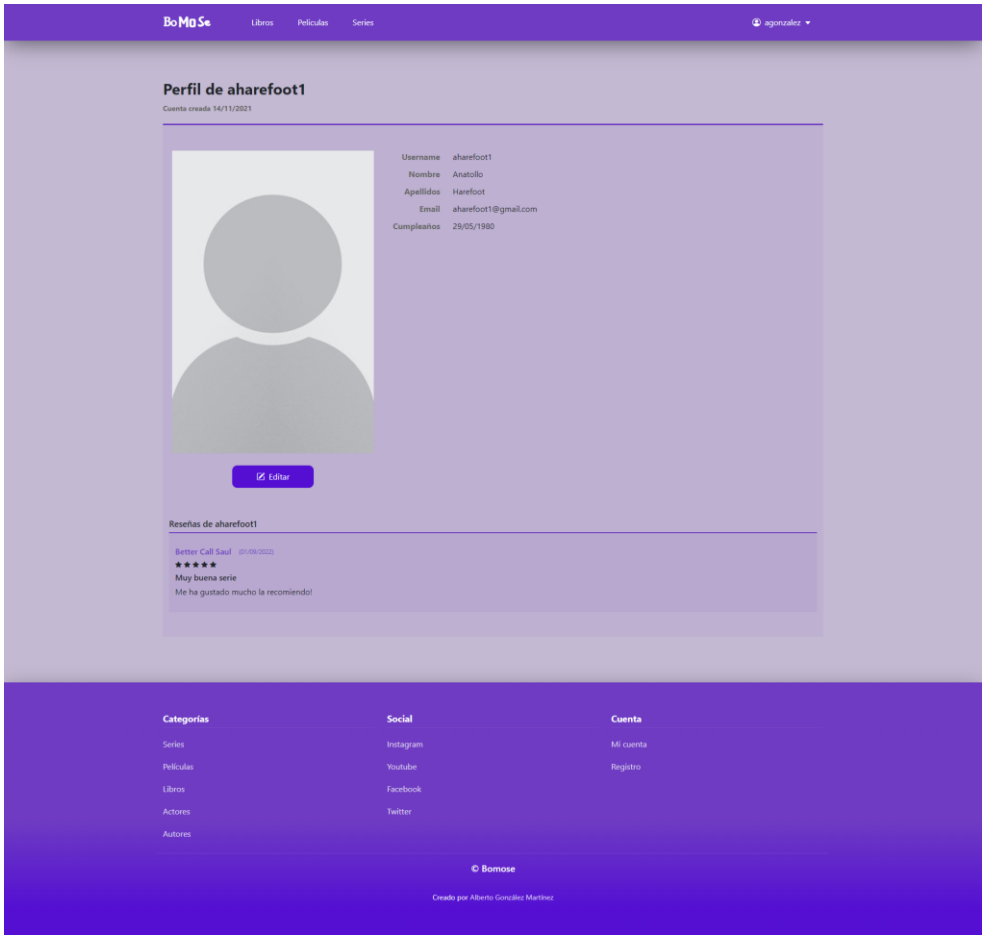
88. Página perfil del usuario registrado

Si el usuario quiere **editar la información de su perfil**, debe hacer click en el botón “Editar”, el cual cuál redirigirá al usuario a la vista para editar la información sobre un usuario.

89. Botón editar información del usuario registrado

5.8. Vista perfil de usuario

En todo momento se podrá **consultar**, la **información** referente a cualquiera de los **usuarios que están registrados en la aplicación**. En esta vista se encuentra la foto de perfil, información sobre el usuario y la lista de reseñas realizadas por este.



90. Página perfil de usuario

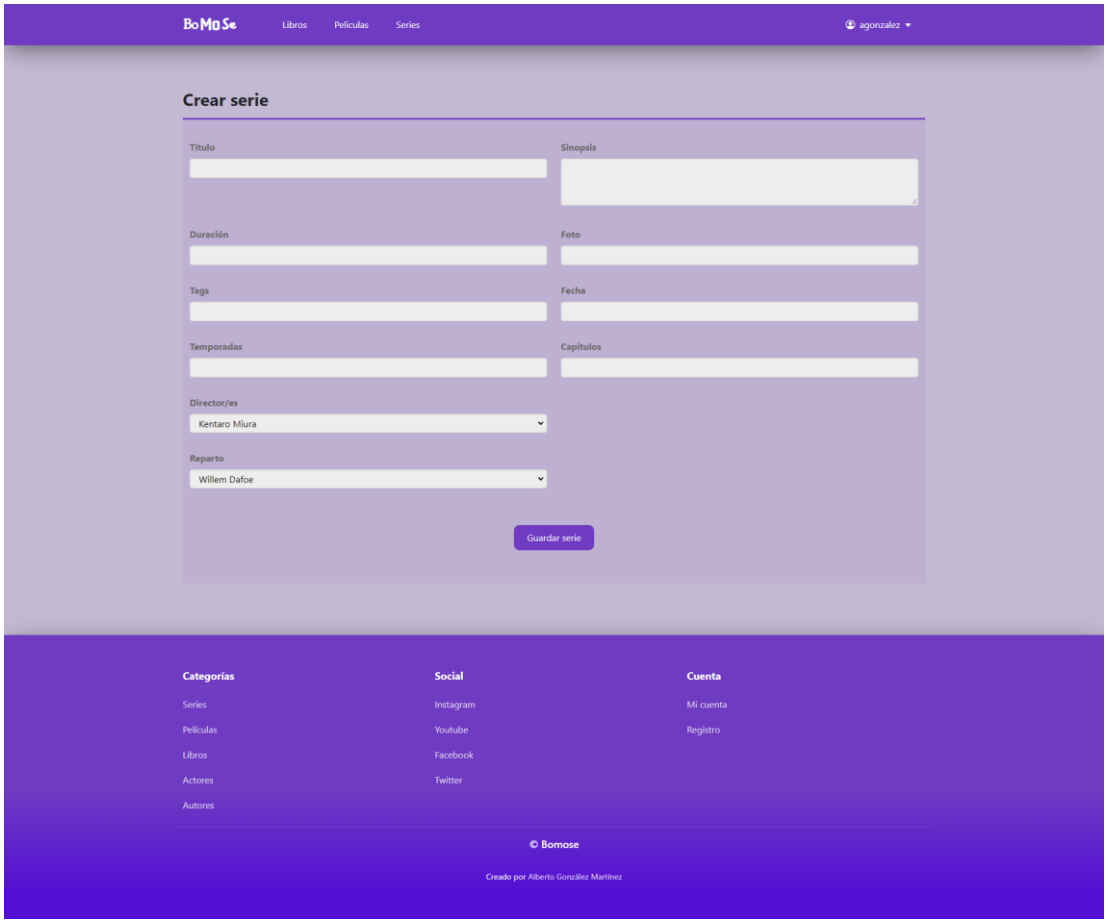
5.8.1. Funcionalidades de administrador

Si el **usuario** es un **administrador**, tendrá la opción de **editar la información** referente al **usuario** haciendo click en el botón “Editar”. Este botón redirigirá al administrador a la vista para editar la información sobre un usuario.

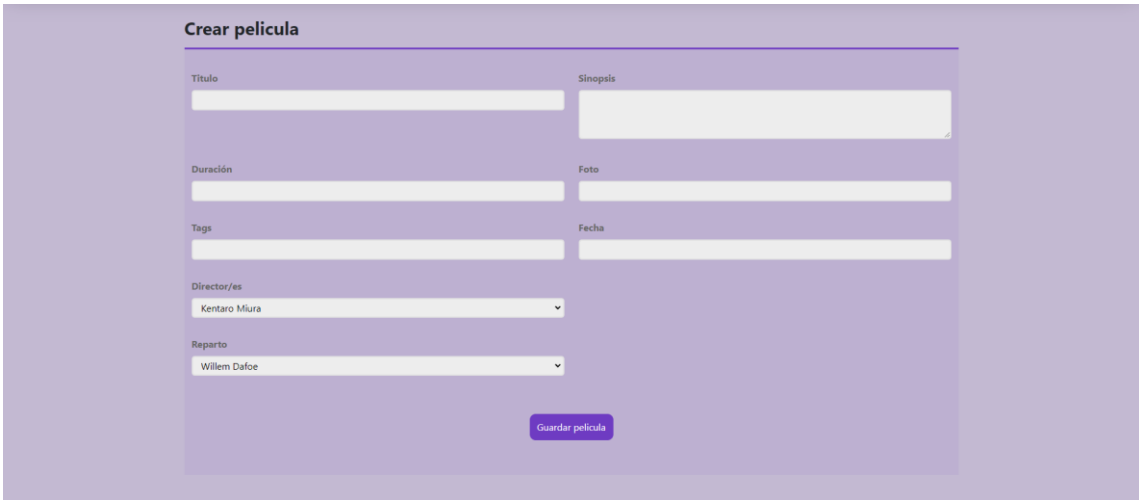
91. Botón editar información de usuario

5.9. Vista crear elemento

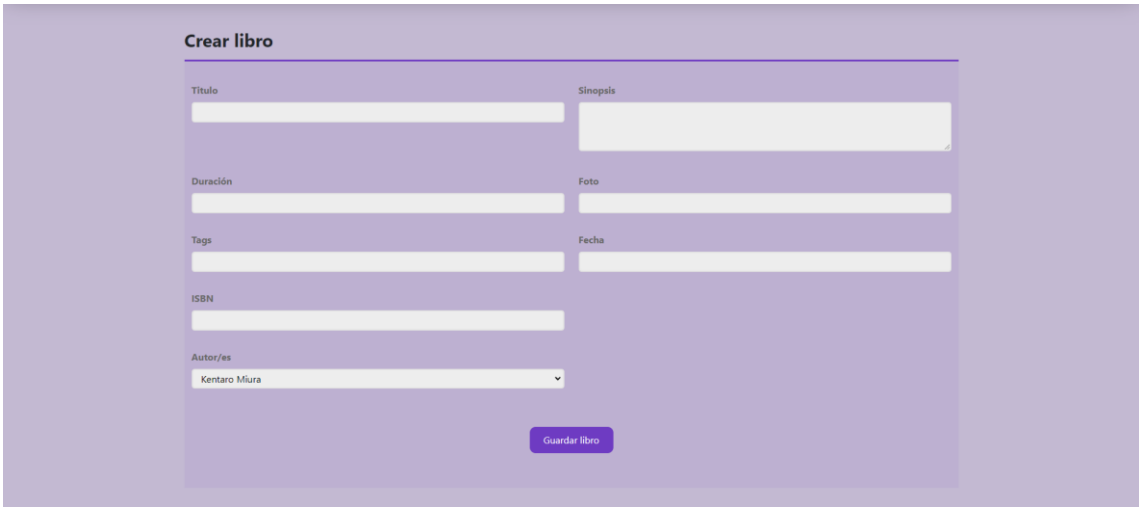
Solo los **administradores de la aplicación** pueden acceder a la vista **para crear una nueva obra.** Hay tres tipos de formularios en esta vista según se quiera crear una serie, película o libro.



92. Página para crear nueva serie



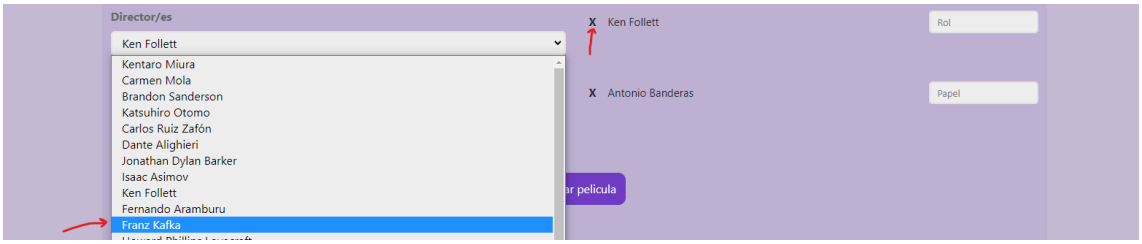
93. Página para crear nueva película



94. Página para crear nuevo libro

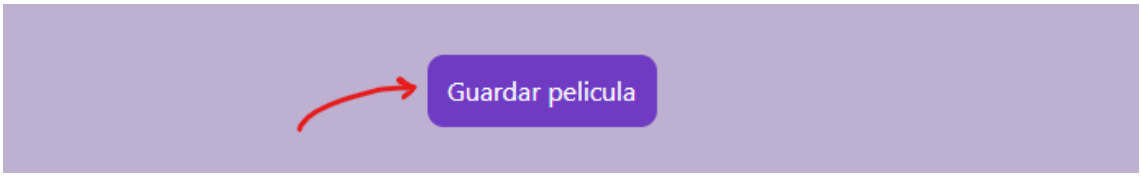
El administrador deberá rellenar los campos del formulario con la información necesaria.

Para **añadir actores o autores al reparto o dirección de la obra,** se deberá hacer click en el campo desplegable y el autor o actor seleccionado, se añadirá a una lista a la derecha del desplegable. En esta lista se podrá borrar un elemento añadido (haciendo click en la X), e indicar el papel o rol que tiene en la obra.



95. Añadir nuevo director a la dirección de una serie

Una vez que se haya introducido la información necesaria, se debe hacer click en el botón “Guardar”.



96. Botón guardar película

Si todo **ha salido correctamente**, aparecerá un mensaje avisando al administrador de que se ha creado el elemento con éxito.

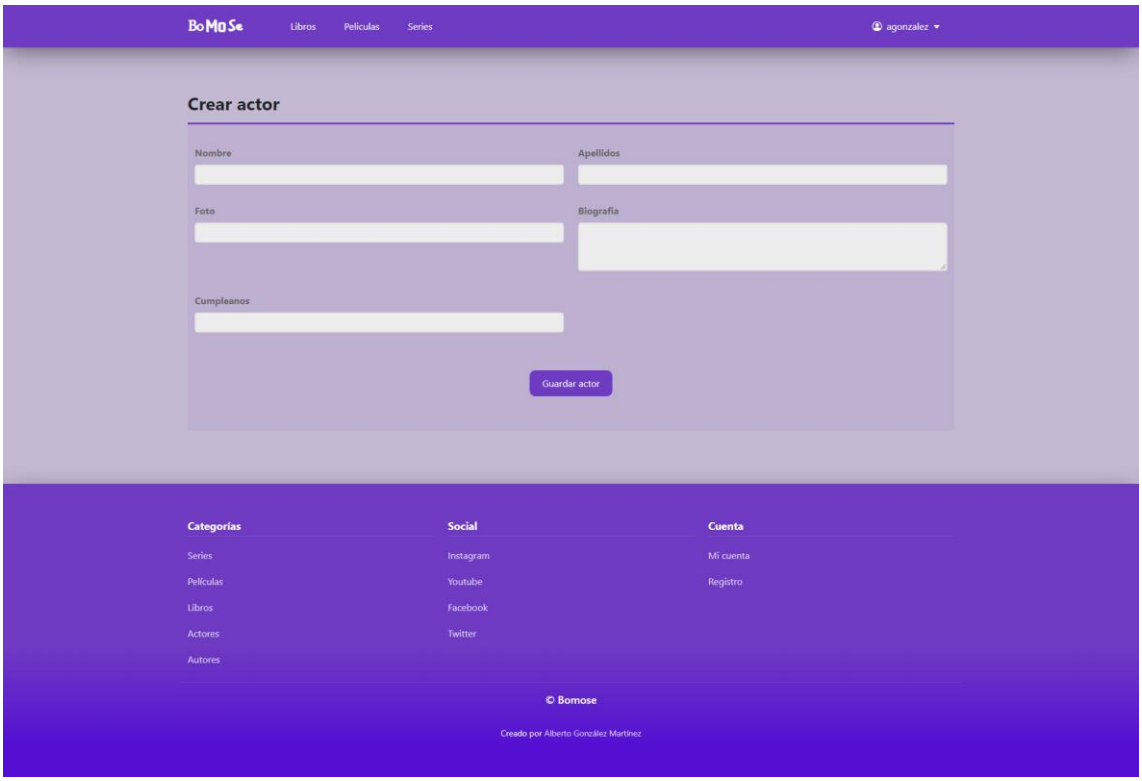
97. Mensaje de confirmación al crear una nueva película

En caso de que **no se haya podido crear la nueva entrada**, se le mostrará al administrador un mensaje de error.

98. Mensaje de error al crear una nueva película

5.10. Vista crear artista

Solo los **administradores** de la aplicación pueden acceder a la vista para **crear un nuevo artista**. Hay dos tipos de formularios en esta vista según se quiera crear una actor o autor.



99. Página para crear un nuevo artista

El administrador deberá rellenar los campos del formulario con la información necesaria.

Una vez que se haya introducido la información necesaria, se debe hacer click en el botón “Guardar”.

100. Botón guardar actor

Si todo **ha salido correctamente**, aparecerá un mensaje avisando al administrador de que se ha creado el elemento con éxito.

101. Mensaje de confirmación al crear un nuevo actor

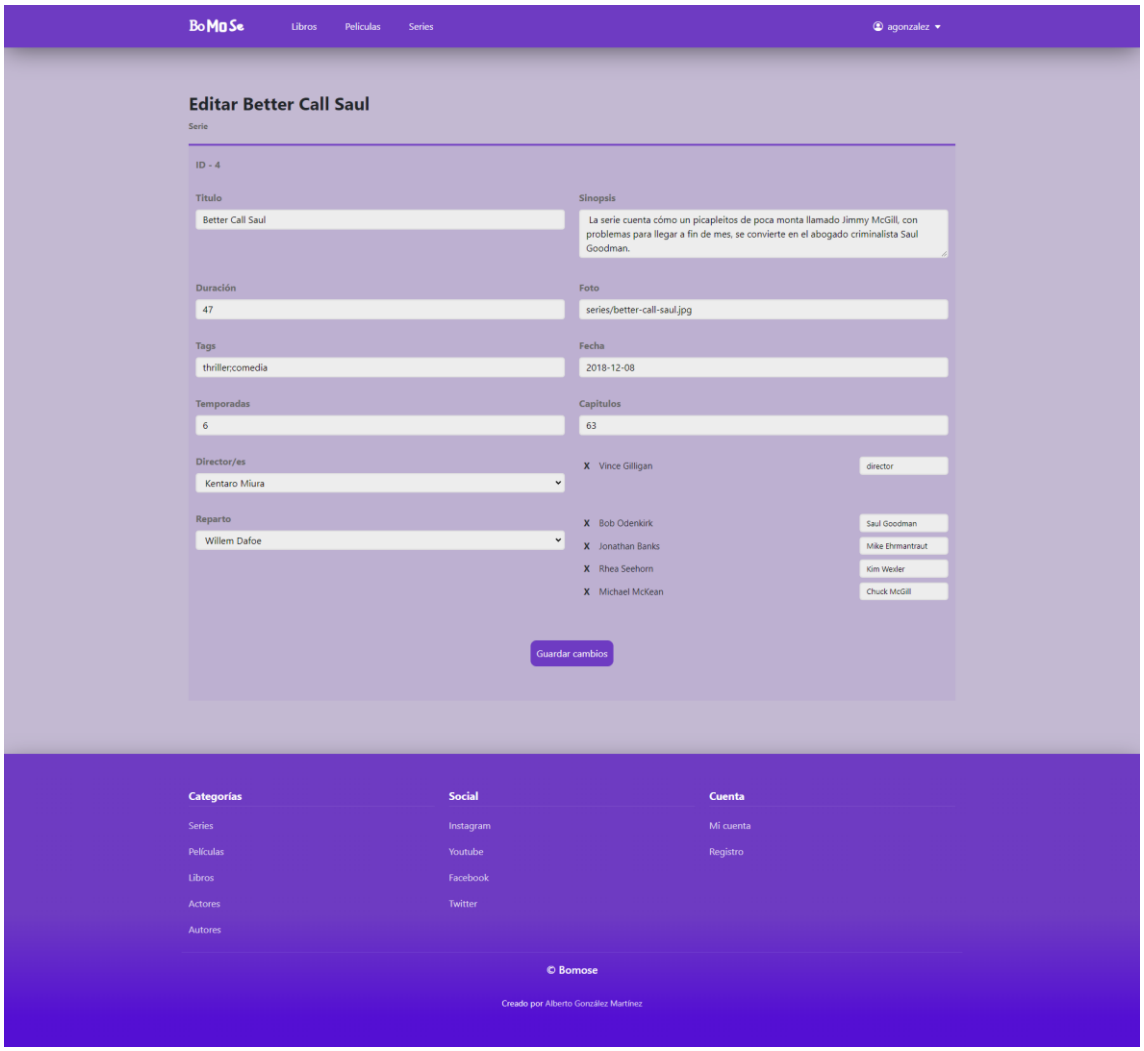
En caso de que **no se haya podido crear la nueva entrada**, se le mostrará al administrador un mensaje de error.

102. Mensaje de error al crear un nuevo actor

5.11. Vista editar película, serie o libro

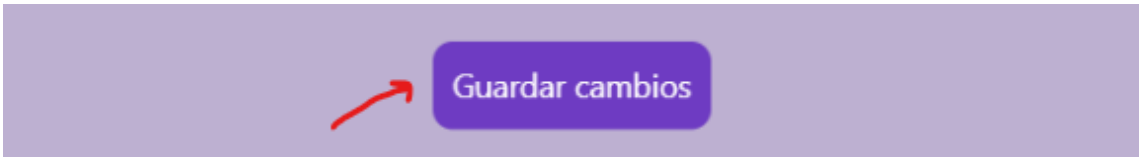
Solo los **administradores** de la aplicación pueden acceder a la vista para **editar la información de una nueva obra**. Hay tres tipos de formularios en esta vista según se quiera crear una serie, película o libro.

Esta vista tiene el mismo formulario que para crear una nueva obra, a diferencia de que, en este caso, el formulario tendrá los campos rellenos con la información actual de la obra.



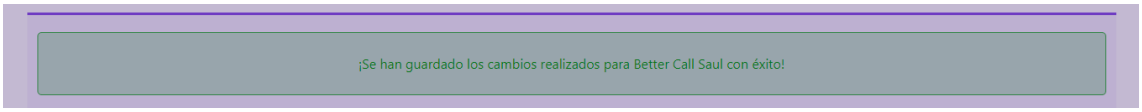
103. Página para editar la información de una serie

Una vez que el administrador haya realizado los cambios requeridos, deberá hacer click en el botón “Guardar cambios”.



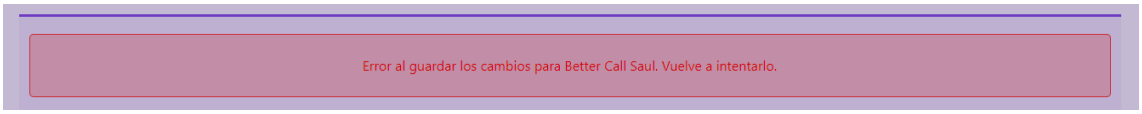
104. Botón guardar cambios realizados sobre una serie, película o libro

Si todo **ha salido correctamente**, aparecerá un mensaje avisando al administrador de que se ha editado la información de la obra con éxito.



105. Mensaje de confirmación al editar una serie

En caso de que **no se haya podido editar la información de la obra**, se le mostrará al administrador un mensaje de error.

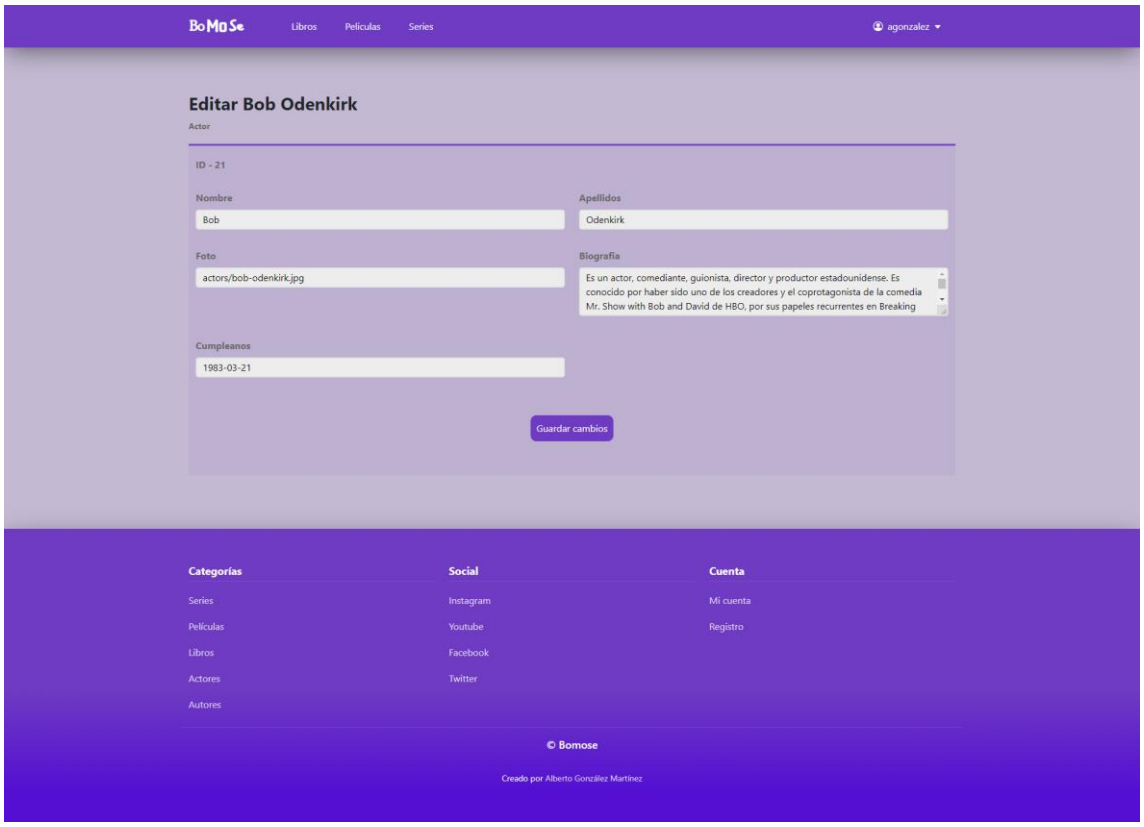


106. Mensaje de error al editar una serie

5.12. Vista editar artista

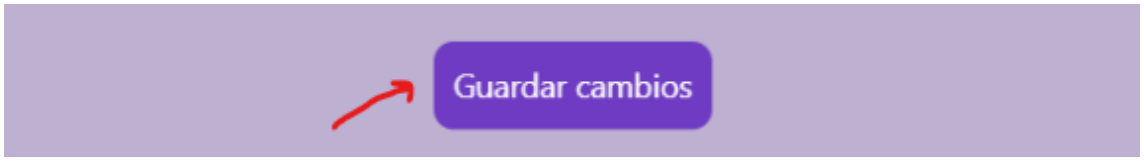
Solo los **administradores** de la aplicación pueden acceder a la vista para **editar la información de un artista.** Hay dos tipos de formularios en esta vista según se quiera crear un actor o autor.

Esta vista tiene el mismo formulario que para crear un nuevo artista, a diferencia de que, en este caso, el formulario tendrá los campos rellenos con la información actual del autor o actor.



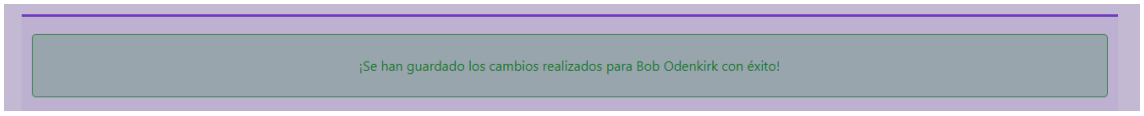
107. Página para editar la información de un artista

Una vez que el administrador haya realizado los cambios requeridos, deberá hacer click en el botón “Guardar cambios”.



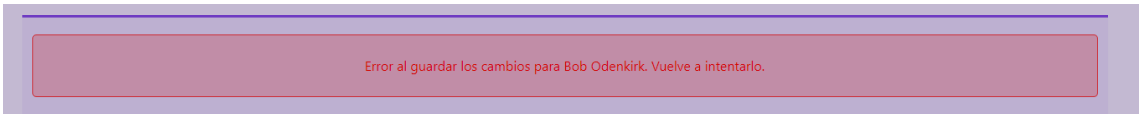
108. Botón guardar las cambios del artista

Si todo **ha salido correctamente**, aparecerá un mensaje avisando al administrador de que se ha editado la información del artista con éxito.



109. Mensaje de confirmación al editar un artista

En caso de que **no se haya podido editar la información del artista**, se le mostrará al administrador un mensaje de error.



110. Mensaje de error al editar un artista

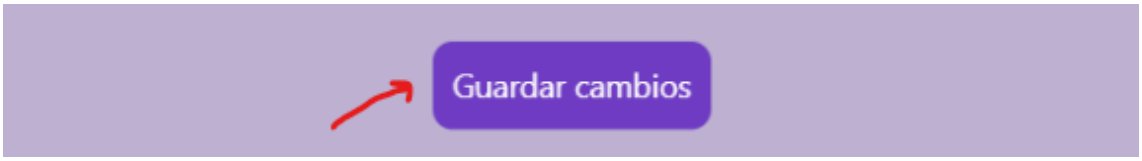
5.13. Vista editar usuario

Solo los **administradores** y el **dueño del perfil** podrán acceder a la vista para **editar la información del usuario**. Esta vista tendrá los campos rellenos con la información actual para el usuario que se quiere editar.

A diferencia de las otras vistas para crear y editar la información, como en este caso, no solo puede acceder un administrador a esta, en esta vista, sí que hay **validación de los datos de los campos del formulario**.

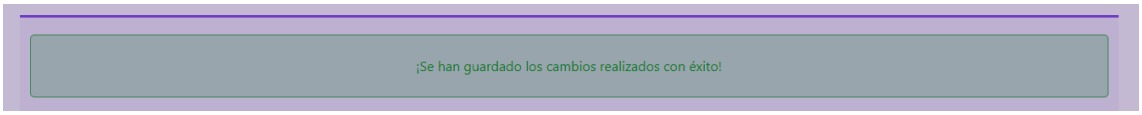
Para poder **cambiar la contraseña**, los dos campos, “Nueva contraseña” y “Repita la contraseña”, tienen que ser iguales.

Una vez que el usuario haya realizado los cambios requeridos, deberá hacer click en el botón “Guardar cambios”.



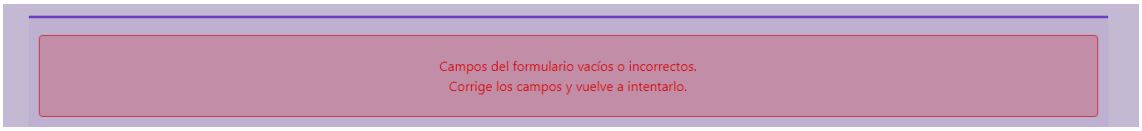
111. Botón guardar los cambios de un usuario

Si todo **ha salido correctamente,** aparecerá un mensaje avisando al usuario de que se ha editado la información con éxito.



112. Mensaje de confirmación al editar un usuario

En caso de que **no se haya podido editar la información**, se le mostrará al usuario un mensaje de error.

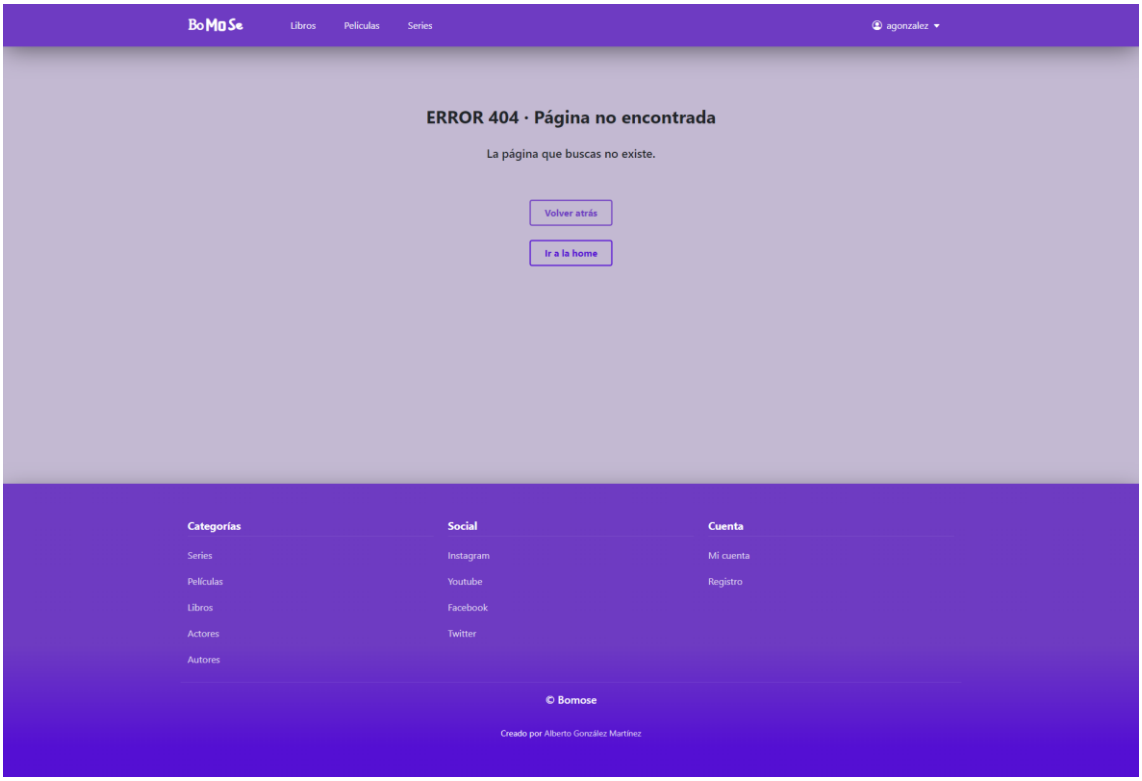


113. Mensaje de error al editar un usuario

5.14. Vista de error

Para **controlar** los posibles **errores que se pueden producir** cuando un usuario navega en la página web, se ha creado una vista que muestra información sobre el error.

En caso de que el usuario intente acceder a una **ruta que no existe**, se mostrará la vista con un aviso de error 404.



114. Página error 404

Si el **recurso** que se intenta cargar **no ha podido ser cargado**, se mostrará una vista con un aviso de error 500.

115. Página error 500

Por último, si el usuario introduce un **parámetro erróneo en la URL** de la página web, se mostrará aviso de error URL.

116. Página error URL

En todas las vistas de error, se le dará la opción al usuario de **volver a la página anteriormente visitada** o **recargar la página** y de ser **redirigido a la home**, al pulsar cualquiera de los botones disponibles en la vista.

6. Manual de despliegue

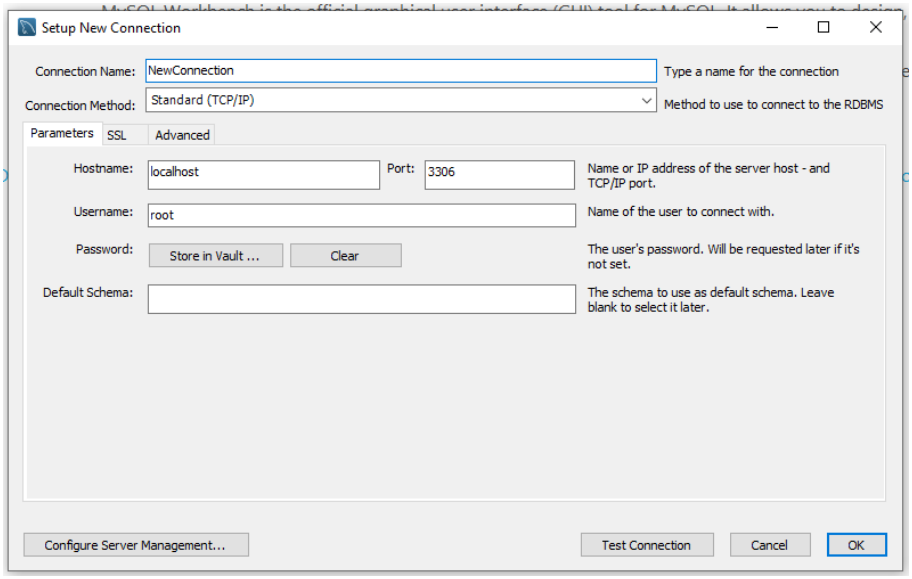
En este apartado se van a exponer los pasos necesarios para realizar el despliegue y prueba del proyecto realizado.

Primer de todo, es necesario **instalar** y **configurar** la **base de datos**. Para ello, se deberá hacer lo siguiente:

1. **Descargar el instalador de MySQL**, y seleccionar, en los pasos del instalador, la versión de desarrollador.

   https://dev.mysql.com/downloads/installer/

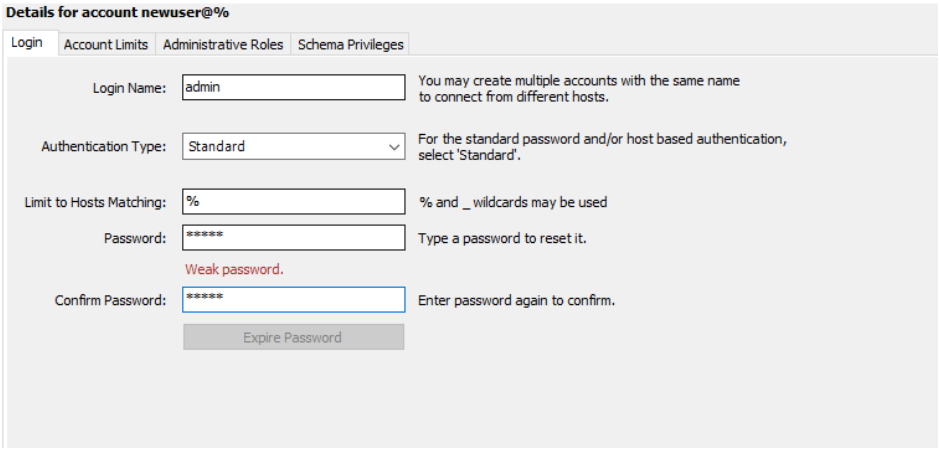
2. **Crear** una **nueva conexión de MySQL** mediante la herramienta MySQL Workbench.



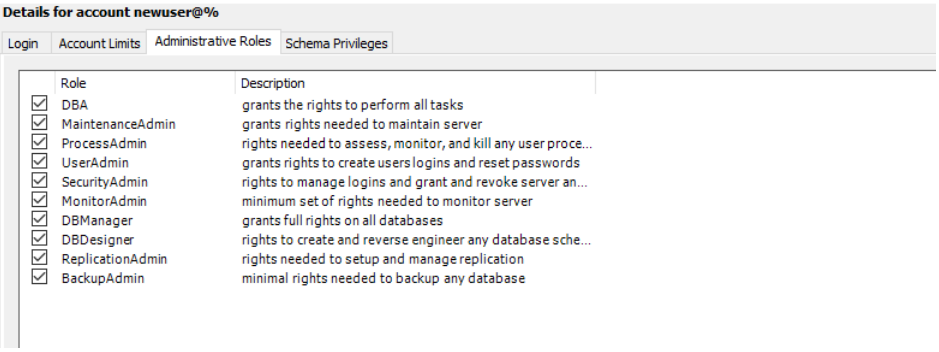
117. Creación de una nueva conexión MySQL

3. **Acceder** a la nueva **conexión creada**.

4. **Crear** un **nuevo usuario**, en la pestaña de “Management”, en la opción de “Users and Privilegies”. El usuario deberá tener el nombre “admin”, contraseña “admin” y todos los permisos.

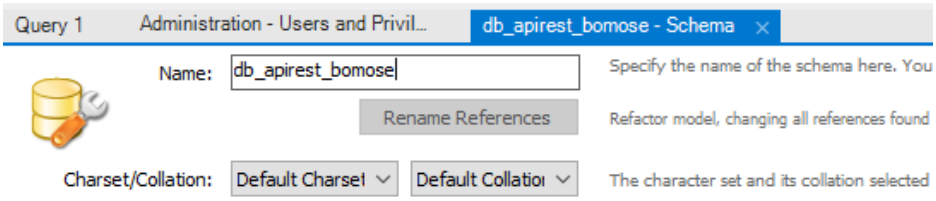


118. Creación de un nuevo usuario en MySQL



119. Roles de usuario en MySQL

5. **Crear** un **nuevo esquema** con el nombre **db_apirest_bomose.**

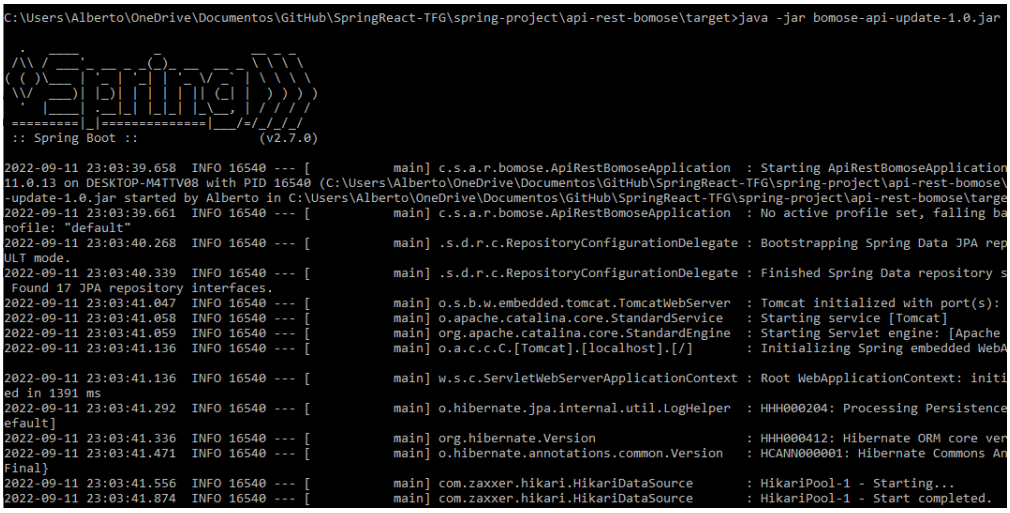


120. Creación de un nuevo esquema en MySQL

Una vez, instalado y configurado el servidor de base de datos MySQL, se deberá **iniciar** el **servidor Spring Boot**, para poner en funcionamiento el API REST. Es importante tener **la versión 11** de java y tener el **puerto 8080 libre**, para que la aplicación pueda ejecutarse correctamente.

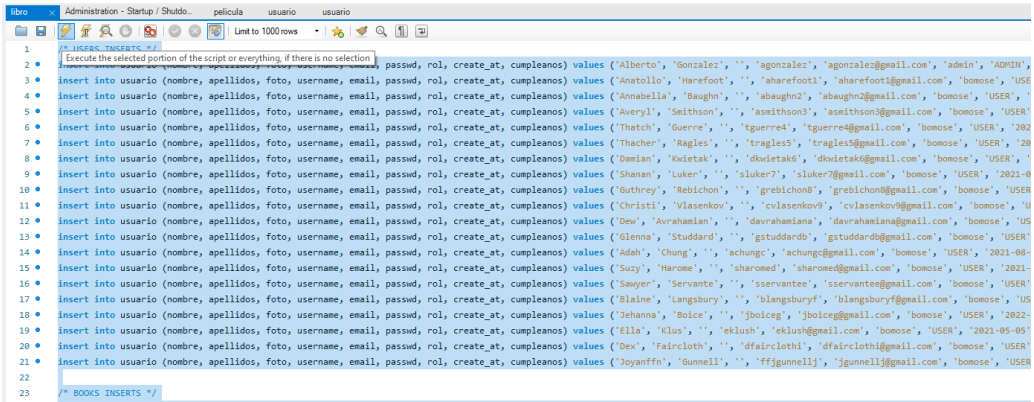
Hay dos formas de iniciar el servidor, con funcionalidad *update* o *create*.

Con **update**, los **cambios** en la base de datos **se guardarán**, si se pausa y se vuelve a iniciar el servidor, para ello se deberá **ejecutar** el archivo con la extensión .jar desde la consola con el comando **"java -jar bomose-api-update-1.0.jar".**



121. Ejecución del archivo update .jar

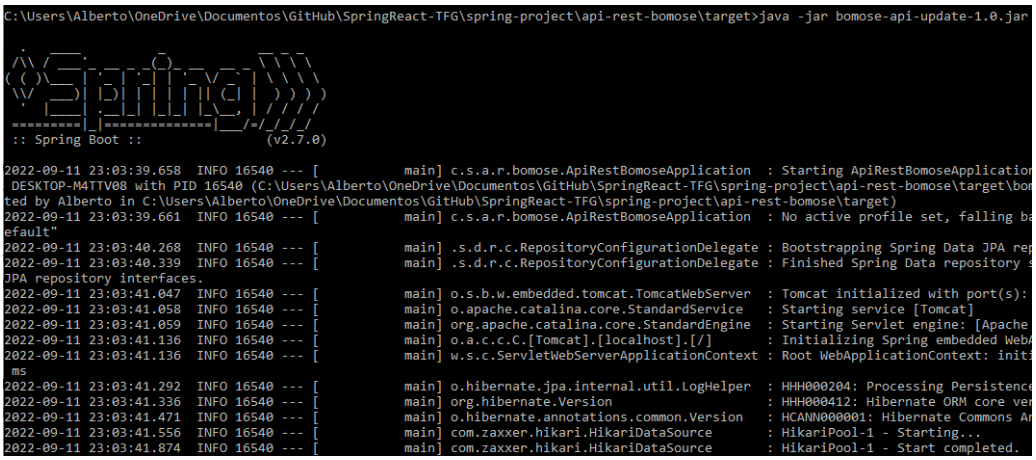
Por otro lado, deberá **ejecutar** el **script con extensión .sql** desde MySQL Workbench, para **poblar la base de datos** con la información necesaria.



122. Ejecución del script SQL para la inserción de datos

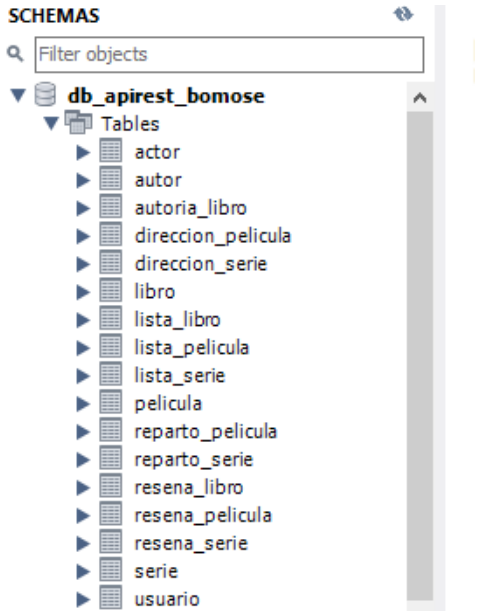
Con **create**, la base de datos se volverá a crear cada vez que se inicie de nuevo el servidor, y, por tanto, **no se guardarán los datos**, pero, en este caso, no será necesario ejecutar el script para insertar la información en la base de datos.

Para iniciar el servidor, se deberá **ejecutar** el archivo con la extensión .jar desde la consola con el comando **"java -Dfile.encoding=UTF8 -jar bomose-api-create-1.0.jar".**



123. Ejecución del archivo create .jar

Una vez iniciada el API REST, en la base de datos, se **crearán todas las tablas** y **relaciones**, dentro del esquema anteriormente creado db_apirest_bomose.



124. Tablas de la base de datos de Bomose en MySQL

Por último, se deberá **iniciar la aplicación de ReactJS** de la siguiente forma:

1. **Abrir** el **proyecto** de react en Visual Studio Code o desde la consola.
2. **Ejecutar** el comando **npm install** desde la carpeta de la aplicación para instalar librerías que podrían no estar instaladas.
3. **Ejecutar** el comando **npm run start**. Es importante que el puerto 3000 esté disponible para que la aplicación de pueda ejecutar sin problemas.

Para poder probar el funcionamiento de la aplicación, se deberá **acceder** a **http://localhost:3000/.** Si se quiere acceder a toda la funcionalidad de la aplicación, habrá que acceder a esta **con permisos de administrador**, para ello, hay que iniciar sesión con el siguiente usuario:

Usuario: agonzalez

Contraseña: admin

Otro **usuario** ya creado en la aplicación **sin permisos de administrador** es:

Usuario: sluker7

Contraseña: bomose

7. Presupuesto

En este apartado se realizará un análisis de los gastos y presupuesto que se deberá destinar o ha sido destinado para implementar el proyecto actual.

7.1. Software utilizado

En el apartado software, todos los **programas** o **herramientas** utilizadas han sido **gratuitas** o se ha conseguido la licencia gratuita por ser estudiante de la Universidad de Alcalá de Henares.

- Java SE Development Kit 11
- Spring Tool Suite 4
- ReactJS
- MySQL Workbench 8.0
- Bootstrap 5.1
- Visual Studio Code
- Modelio
- Postman
- Repositorio GitHub
- Microsoft Word

7.2. Formación y cursos

Para aprender a usar Spring Boot se ha comprado y completado uno de los **cursos** disponibles en la plataforma de cursos online Udemy.

El curso que se ha utilizado es el siguiente, Spring Framework 5 & Spring Boot 2 desde cero a experto 2022 (https://www.udemy.com/course/spring-framework-5/)

Recurso	Precio
Curso Udemy	15€
TOTAL	15€

7.3. Servicio hosting

Una vez completado el desarrollo del proyecto, se necesitará realizar la compra o alquiler del dominio web, alquiler de un servicio hosting y compra del certificado SSL para mostrar la página web como página segura.

Para el servicio hosting de la web, se utilizará la Amazon Web Service, en donde se podrán alojar la base de datos, el API y la aplicación React. Además, se usará un bucket S3 para guardar los archivos estáticos de la aplicación, como, por ejemplo, las fotos.

Recurso	Precio
Dominio web y correo	14€ / año
Servicio hosting AWS	161€ / año
Certificado SSL	25€ / año
TOTAL	200€ / año

8. Conclusiones

Como se ha expuesto a lo largo de la memoria, el interés del contenido multimedia ha ido creciendo a la largo de los años, por ello, Bomose **ayudará**, a los **consumidores de ese contenido**, a **encontrar la mejor opción** basada en sus gustos y las recomendaciones de otros usuarios, haciendo que estos usuarios puedan ahorrar tiempo a la hora de elegir que ver o leer.

Bomose pretende **llegar y ayudar a cualquier tipo de público** que esté interesado en leer libros y ver películas o series, por ello, **no** tiene **franja de edad** o tipo de usuario concreto, ya que el catálogo de series, películas o libros que se pretende conseguir va a ser amplio y variado. Esto también está reflejado en la **interfaz** de la página web, que pretende ser lo más **simple** y **fácil** posible, para que pueda ser accesible por todo tipo de público y navegable en cualquier dispositivo, debido que esta interfaz ha sido desarrollada y pensada para tener un diseño *responsive*.

En el **apartado tecnológico**, este proyecto se ha desarrollado con tecnologías actuales, ampliamente usadas y con una gran compatibilidad, con lo que, la actualización, la implementación de nuevas funcionalidades y la evolución de la aplicación será más sencilla y necesitará de menos recursos y tiempo.

Finalmente, una vez explicadas la idea principal y las características de la aplicación en esta memoria, las **líneas de trabajo futuro** son ir añadiendo **nuevas funcionalidades y características** a la página web, además de **mejorar** las que están **actualmente desarrolladas**. Algunas de las nuevas funcionalidades que se quieren añadir en un futuro son:

- Ampliar el catálogo de películas, series, libros, autores y actores.
- Más seguridad para los usuarios.
- Subir fotos desde la página web.
- Buscador para el contenido de la web.
- Incluir iframe en la información de un elemento para tráileres o vídeos de interés.

Otra de la líneas de trabajo futuro importantes, es valorar las distintas **opciones de monetización** de las que se disponen, para poder cubrir los costes del desarrollo y los que genera la página web de forma anual. Algunas de las opciones podrían ser, incluir anuncios en la página web, habilitar un link para que los usuarios puedan realizar contribuciones voluntarias o idear algún tipo de membresía o contenido premium.

9. Glosario de términos

API REST: es una interfaz de programación de aplicaciones que define un conjunto de reglas mediante las cuales las aplicaciones y dispositivos se conectan y comunican entre sí y REST

Aspect ratio: también llamado ratio o relación de aspecto, es la proporción que hay entre el ancho y alto de una imagen.

Back-end: en desarrollo web back-end se denomina a la parte lógica de una página web que hace posible su funcionamiento, en el caso de este proyecto, el API REST es parte del back-end de este.

Clases: en programación un clase es un conjunto de atributos y métodos que definen o representan una entidad, objeto o concepto.

Contenedor: es una estructura web que abarca un recuadro y puede colocarse en cualquier parte de una página, en esta estructura se puede insertar contenido HTML.

Controlador: dentro del patrón modelo vista controlador, es el encargado de operar y proporcionar los datos al usuario.

CRUD: es el acrónimo de "Crear, Leer, Actualizar y Borrar" y se usa para referirse a las operaciones básicas que se realizan en una base de datos.

Display: es un término inglés, que, en lenguaje web, es la forma en la que se muestra o visualiza un elemento. Por ejemplo, display none, se usa para cuando no se quiere mostrar un elemento.

Endpoint: referido a un API, un endpoint es el extremo de una conexión donde se recibe la llamada al API.

Entidades: representa un objeto o cosa, dentro de una base de datos, se refiere a las tablas que la componen.

Escalabilidad: es la capacidad que tiene, en este caso, una aplicación o software, para adaptarse y crecer.

Etiqueta: son fragmentos de código que representan un elemento HTML.

Footer: también conocido como pie de página, es la parte inferior de una página web.

Framework: es un término inglés, también conocido como marco de trabajo, es un estructura o plantilla que sirve para elaborar la base de un proyecto con objetivos específicos.

Front-end: es la parte visible de cara al usuario, transforma y presenta los datos para que el usuario pueda ver e interactuar con estos.

Full-stack: es un término que aúna el desarrollo back-end y el desarrollo front-end de una página web.

Grid: es un término inglés que se traduce como rejilla o cuadrícula. En Bootstrap, el grid representa la división de un contenedor HTML por columnas.

Hosting: es un servicio de almacenamiento en línea para que los recursos, estén disponibles y accesibles vía Internet.

HTTP: el Protocolo de Transferencia de HiperTexto, es un protocolo que permite la transferencia de información en línea.

IDE: es una aplicación que proporciona a los desarrolladores un entorno de programación que facilita el desarrollo de código de una manera eficiente.

Interfaz: es la parte visual de una aplicación o software, mediante la cual el usuario y la aplicación interaccionan.

Maquetación: en lenguaje web, la maquetación es el proceso de transformar un idea o diseño en una interfaz funcional mediante la escritura de código, en este caso HTML, CSS y JS.

Margin: es una propiedad CSS, se refiere al margen o espacio que tiene un elemento HTML respecto a otros.

Media Queries: en desarrollo web, es una funcionalidad de CSS que permite adaptar las propiedades o características de un elemento, en función de las características del dispositivo, por ejemplo, el ancho o alto de la pantalla.

Modularidad: es una propiedad que permite dividir una aplicación en componentes, los cuales pueden ser reutilizados.

Open Source: también conocido como código abierto, es un término que se refiere al software que tiene un código el cual es accesible, modificable y distribuible por los usuarios.

Padding: es una propiedad CSS, el espacio que tiene un elemento hasta los margen de la etiqueta que lo contiene.

Parámetro: dentro de una URL, un parámetro, es una forma de transferir información.

Props: en ReactJS, se denomina prop, a las propiedades que reciben los componentes, es una forma de personalizar y pasar información a estos.

Renderizar: es la acción que realiza, por ejemplo, el navegador, cuando al entrar a una página web, el contenido es dibujado en la pantalla para que el usuario pueda verlo.

Responsive: en desarrollo web, el término responsive hace referencia al contenido que es capaz de adaptarse a cualquier dispositivo sin importar las dimensiones de este.

Script: es un documento que contiene instrucciones escritas en un lenguaje de programación que tienen como objetivo realizar diferentes tareas.

SEO: son las siglas de Search Engine Optimization y se refiere a las acciones que están orientadas a conseguir un mejor posicionamiento de un sitio web en las búsquedas de Google.

Streaming: es la tecnología que permite poder ver y oír contenido multimedia a través de internet sin necesidad de tener que descargar los recursos necesarios.

URL: son las siglas de Uniform Resource Locator (Localizador de Recursos Uniforme) y se refiere a la dirección única que tiene un recurso y que se usa para poder acceder a este.

Usabilidad: define la facilidad o la sencillez que tiene un aplicación para ser utilizada por un usuario.

10. Bibliografía

- Documentación de Spring Boot. Spring Boot Reference Documentation. https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ [12/09/2022]

- Documentación de Bootstrap. Get started with Bootstrap · Bootstrap v5.2. https://getbootstrap.com/docs/5.2/getting-started/introduction/ [12/09/2022]

- Documentación de ReactJS. Empezando – React. https://es.reactjs.org/docs/getting-started.html [12/09/2022]

- Documentación de MySQL. MySQL :: MySQL Documentation. https://dev.mysql.com/doc/ [12/09/2022]

- Documentación de React Slick. React Slick Documentation. https://react-slick.neostack.com/docs/get-started/ [12/09/2022]

- Documentación de react-router-dom. React Router: Declarative Routing for React.js. https://v5.reactrouter.com/web/guides/quick-start [12/09/2022]

- Ordenar un array en javascript. Javascript - Sort an array of objects in React and render them - Stack Overflow. https://stackoverflow.com/questions/43572436/sort-an-array-of-objects-in-react-and-render-them [12/09/2022]

- Cambiar el estado de un componente padre desde un componente hijo en ReactJS. How to set Parent State from Children Component in ReactJS? – GeeksforGeeks. https://www.geeksforgeeks.org/how-to-set-parent-state-from-children-component-in-reactjs/ [12/09/2022]

- Configuración de rutas absolutas en ReactJS. Why and How to Use Absolute Imports in React. https://javascript.plainenglish.io/why-and-how-to-use-absolute-imports-in-react-d5b52f24d53c [12/09/2022]

- Hacer una redirección con la librería react-router-dom. Javascript - React: 'Redirect' is not exported from 'react-router-dom' - Stack Overflow. https://stackoverflow.com/questions/63690695/react-redirect-is-not-exported-from-react-router-dom [12/09/2022]

- Animación de carga en CSS. Apple Loader CSS - iPhone Loader. https://codepen.io/serkanzturk/pen/wvmGVMb [12/09/2022]

- Ventajas de Spring Boot. Por qué DEBES usar Spring Boot en tus proyectos JAVA - //Arteco. https://www.arteco-consulting.com/post/por-que-debes-usar-spring-boot [12/09/2022]

- ¿Qué es Postman? | lamadriguerabit.com. https://lamadriguerabit.com/articulos/que-es-postman/ [12/09/2022]

- Qué es React y cómo funciona. React | Qué es, para qué sirve y cómo funciona | Descúbrelo todo. https://tech.tribalyte.eu/blog-que-es-react [12/09/2022]

- Ventajas de usar ReactJS. https://sistemasgeniales.com/software/las-10-ventajas-principales-de-usar-react-js/ [12/09/2022]

- Ventajas de las Single Page Applications. Ventajas de crear tu web con Single Page Applications (SPAs) - Making Science. https://www.makingscience.es/blog/ventajas-de-crear-tu-web-con-single-page-applications-spas/ [12/09/2022]

- ¿Qué es Bootstrap? Bootstrap 4: Qué es, cómo instalarlo en tu web y cómo se utiliza. https://raiolanetworks.es/blog/bootstrap/#que_es_bootstrap [12/09/2022]

- Ventajas de MySQL. Las Ventajas de MySQL por Sobre Otras Bases de Datos. https://www.hn.cl/blog/las-ventajas-de-mysql-por-sobre-otras-bases-de-datos/#:~:text=La%20cualidad%20m%C3%A1s%20destacada%20por,%2DF%C3%A1cil%20de%20usar. [12/09/2022]

- Taxonomía de requisitos | Marco de Desarrollo de la Junta de Andalucía. https://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/408 [12/09/2022]