Facultad de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
EN INGENIERÍA INFORMÁTICA
MENCIÓN EN INGENIERÍA DEL SOFTWARE

Aplicación para puntuar y llevar un seguimiento de las series y películas del catálogo de los principales servicios de streaming

Estudiante: José Julio Villanueva Rodríguez
Dirección: Fernando Bellas Permuy

A Coruña, setembro de 2021.

A mis padres y a mi hermano, por toda la confianza y paciencia que tuvieron conmigo.

Agradecimientos

Para empezar, agradecer a mi familia todo el esfuerzo que hicieron para que pudiese llegar hasta aquí. A mi padre por ser un ejemplo a seguir, por su perseverancia y por su envidiable humildad. A mi madre por su manera de darme una visión diferente de las cosas y siempre mostrarme otros puntos de vista, su inquebrantable fe en mi y por heredarme su pasión por la lectura. A mi hermano, con quien tanto discuto, por todas las veces que me aguantó y tuvo que escucharme, por siempre intentar sacar la mejor versión de mi y actuar de *Pepito Grillo* cuando era necesario.

A mis amigos, sin los cuales este camino se hubiese hecho mucho más duro. A los del club, por estar ahí en todas las competiciones de día y de noche. A los de la facultad, por toda la ayuda que me dieron y las veces que tuvieron que hacer de *profesores* conmigo.

A Fernando, por toda la paciencia y los consejos que me dio a lo largo del proyecto, además de enseñarme el campo de la informática del cual estoy enamorado, el desarrollo web.

Resumen

La finalidad del proyecto es proporcionar una aplicación web a los usuarios de los principales servicios de streaming (Netflix, HBO, Disney +, Movistar + o Prime Video) para que puedan llevar un seguimiento de las películas y series que vean, puntuarlas y comentarlas de forma que los usuarios puedan interactuar entre ellos.

El *Backend* de la aplicación está desarrollado en Java EE usando el *Framework* de Spring Boot. Para la persistencia de datos usamos JPA y MySQL. Para la comunicación entre el *backend* y el *Frontend* de la aplicación se recurre a una API REST. El *frontend* consiste en una aplicación Single Page Application (SPA) implementada en Javascript, usando los frameworks React y Redux.

Estos servicios de streaming no ofrecen ninguna API para poder consultar su catálogo. Sin embargo, existen APIs de terceros que sí lo hacen, aunque en general, no de forma gratuita. En el proyecto se usa una API que da información sobre el catálogo de Netflix de manera gratuita, pero limita el número de consultas diarias. El modelado de datos utilizado en el presente trabajo es independiente del servicio de streaming.

Abstract

The purpose of the project is to provide a web application to users of major streaming services (Netflix, HBO, Disney +, Movistar + or Prime Video) so they can keep track of the movies and series they watch, rate them and comment on them so that users can interact with each other.

The backend of the application is developed in Java EE using the Spring Boot framework. For data persistence we use JPA and MySQL. For the communication between the application's *backend* and *frontend* we use a REST API. The *frontend* consists of a SPA application implemented in Javascript, using the React and Redux frameworks.

These streaming services do not offer any API to query their catalog. However, there are third-party APIs that do, although in general, not for free. The project uses an API that gives information about the Netflix catalog for free, but limits the number of daily queries. The data modeling used in this work is independent of the streaming service.

Palabras clave:

- Aplicación SPA
- Java EE
- Spring Boot
- React
- Redux
- API REST
- MySQL

Keywords:

- SPA Application
- Java EE
- Spring Boot
- React
- Redux
- REST API
- MySQL

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introdución

En 1997, pleno auge de los videoclubs, nace en California la empresa Netflix, que revolucionaría el mercado del alquiler de películas. A diferencia de sus competidores, dicho alquiler no se realizaba en un establecimiento físico, sino que el servicio se ofrecía a través de correo postal. Aunque en sus primero años empezó con pérdidas, la empresa aguantó y no fue hasta 2003 que empezó a tener ganancias. La empresa quería seguir creciendo y en 2007 cuando cambiarían para siempre el mercado del alquiler de DVDs.

Buscando expandirse por Latinoamérica, llegaron a la conclusión de que el servicio de envío por correo postal de DVDs no era factible a escala global. Aprovechando el rápido avance tanto de internet como del número de ordenadores personales, decidieron ofrecer su servicio en formato Video On Demand (VOD).

Han pasado 14 años e indiscutiblemente Netflix se ha ganado el sobrenombre de "the global internet TV network". En abril de este año, Netflix tuvo más de 200 millones de suscripciones (que no de usuarios únicos) y a esta se le suman otras empresas de streaming como HBO (140 millones de suscripciones), Disney+ (100 millones), Amazon Prime Video (200 millones) y Movistar+.

Sin lugar a dudas, el formato VOD es el más popular.

1.1 Contexto

Actualmente Netflix tiene un catálogo de más de 4800 títulos, por lo que si no se tiene muy claro que película o serie ver, puede que se tenga que gastar tiempo buscando en su catálogo. Para que esto no ocurra, Netflix tiene un algoritmo que recomienda películas en base a los gustos del usuario, ya que permite valorar positiva o negativamente las películas y series que este ve.

Este algoritmo cumple su función: recomienda películas que, basándose en las valoraciones del usuario, le deberían gustar. Pero esto tiene un inconveniente: no se puede saber si

una película es "buena" o "mala". Una película puede compartir similitudes con otras que le gustasen al usuario y ser "mala". Aunque esto sea una valoración subjetiva, normalmente la crítica del público suele ser bastante acertada, por lo que muchas veces el voto de la gente es suficiente para intuir si una película valdrá la pena o no.

Además, Netflix no tienen ningún tipo de interacción entre los propios usuarios, por lo que muchas veces si se quiere saber si una película o serie merecerá la pena, se deberán mirar críticas y valoraciones de ella en webs externas.

Esta problemática también se extiende en mayor o menor medida a otros servicios de streaming como HBO, Disney +, Movistar + o Prime Video.

1.2 Objetivos

El objetivo del proyecto es proporcionar a los usuarios una plataforma web destinada a la crítica y valoración de las series y películas de las principales empresas de servicio de streaming.

Los usuarios podrán puntuar las series y películas, participando así en su valoración global. También podrán escribir comentarios en los cuales den su opinión acerca de la película o para mantener una conversación con otro usuario sobre algún capítulo. A su vez, los comentarios también podrán ser puntuados por los propios usuarios, demostrando su conformidad o inconformidad con este.

1.3 Visión global del sistema

El proyecto consistirá en un aplicación web desarrollada con JavaScript, usando los frameworks React y Redux para gestionar el estado.

Para conectarse con el *backend*, el *frontend* consumirá una API REST desarrollada con el framework Spring Boot. La aplicación también tendrá una base de datos relacional MySQL para guardar tanto la información de los usuarios, como la de los comentarios, votos e ítems del catálogo.

La información de los ítems del catálogo la se obtendrán gracias a una API externa a los servicios de streaming, ya que ninguno de estos proporcionan una API que nos ofrezca información sobre su catálogo. La API usada en cuestión es uNoGS, que de forma gratuita nos ofrece información sobre las películas y series que se usarán para poblar la base de datos.

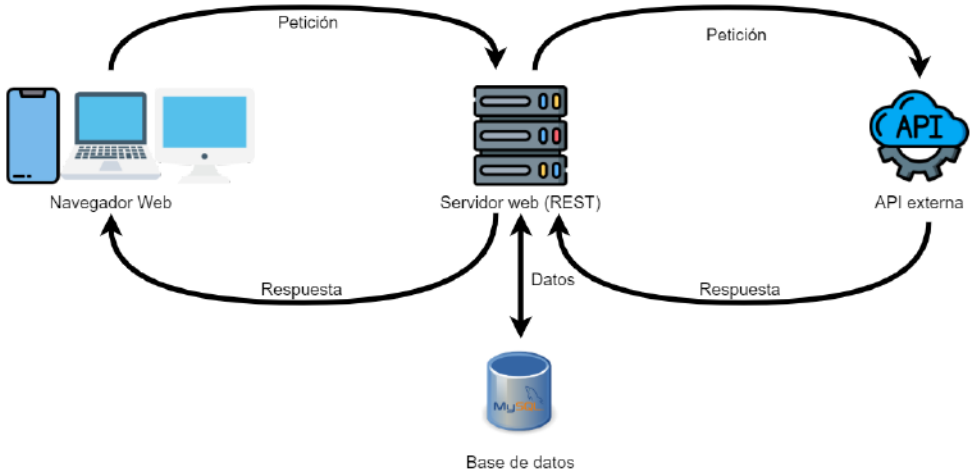

Figura 1.1: Arquitectura del sistema

Como se puede ver en la figura 1.1, la aplicación web se comunicará con el servidor de la aplicación a través de una API REST y el servidor devolverá la respuesta a la petición. A su vez, el servidor web se comunicará con la API externa que se usará para poblar la base de datos. Estas peticiones se harán periódicamente cada 24 horas, en donde se solicitarán los ítems añadidos desde que se realizó la última petición. Esta información será guardada en una base de datos de MySQL.

Este modelo de datos es independiente del servicio de streaming y del API que se use, ya que se desarrolló teniendo en cuenta el posible uso de otras APIs de cara a tener un catálogo más completo.

Capítulo 2

Estado del arte

Hoy en día hay infinidad de páginas web que ofrecen servicios parecidos a los de este proyecto. Sin embargo, ninguna llega a tener todas las características que la aplicación realizada tiene, aunque en algunas coincidan.

2.1 IMDb

IMDb [1] es la página líder en información sobre películas y series. Es una aplicación que lleva desde los años 90 en funcionamiento. Al principio servía como base de datos para películas, pero con el tiempo ha ido aumentando sus servicios hasta el día de hoy, que llega a tener un flujo de 100 millones de usuarios únicos mensuales.

Al igual que la aplicación desarrollada, IMDb ofrece información sobre series y películas de los principales servicios de VOD. También permite dejar críticas sobre películas, series e incluso episodios concretos (a diferencia de nuestro proyecto que también lo permite en las temporadas).

Algo que diferencia la aplicación realizada para este proyecto de IMDb, es la sencillez de la interfaz de usuario. IMDb, al tener tantas funcionalidades, llega a tener una interfaz que en algunos momentos puede llegar a ser caótica y hacer que el usuario se pierda (figura 2.1). No obstante, la interfaz de la aplicación desarrollada es muy sencilla, rápida e intuitiva de usar.

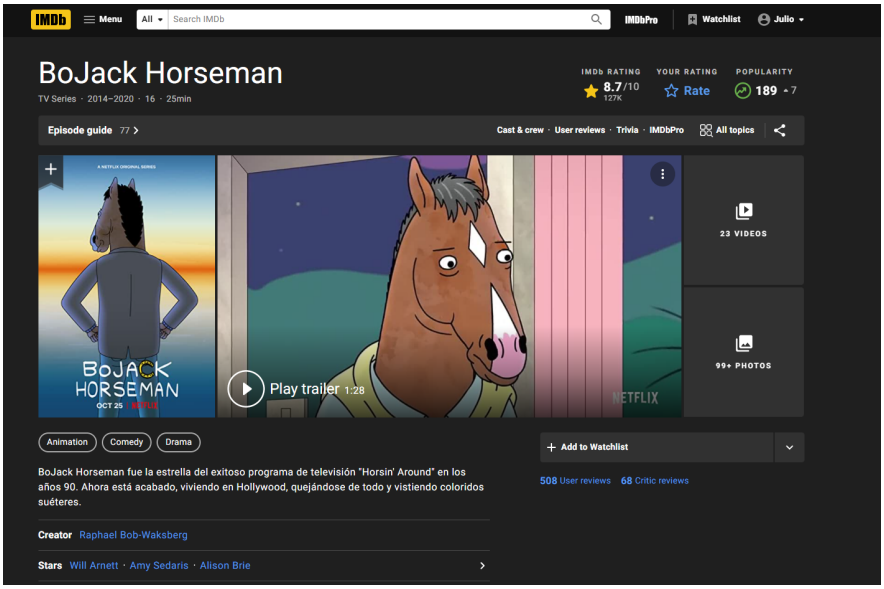

Figura 2.1: Detalles de una serie en IMDb

2.2 MyAnimeList

MyAnimeList [2] es una página web parecida a IMDb, con la diferencia de que se centra exclusivamente en series y películas de animación.

Esta aplicación permite la publicación de críticas en las películas, series y episodios, pero no en las temporadas. También cuenta con un sistema de seguimiento parecido al de la aplicación desarrollada, en el que al marcar películas como vistas, después se pueden ver diferentes estadísticas relacionadas con el tiempo de visualización.

Al igual que le pasaba a IMDb, su interfaz puede llegar a ser liosa hasta que el usuario se acostumbre a ella (figura 2.2).

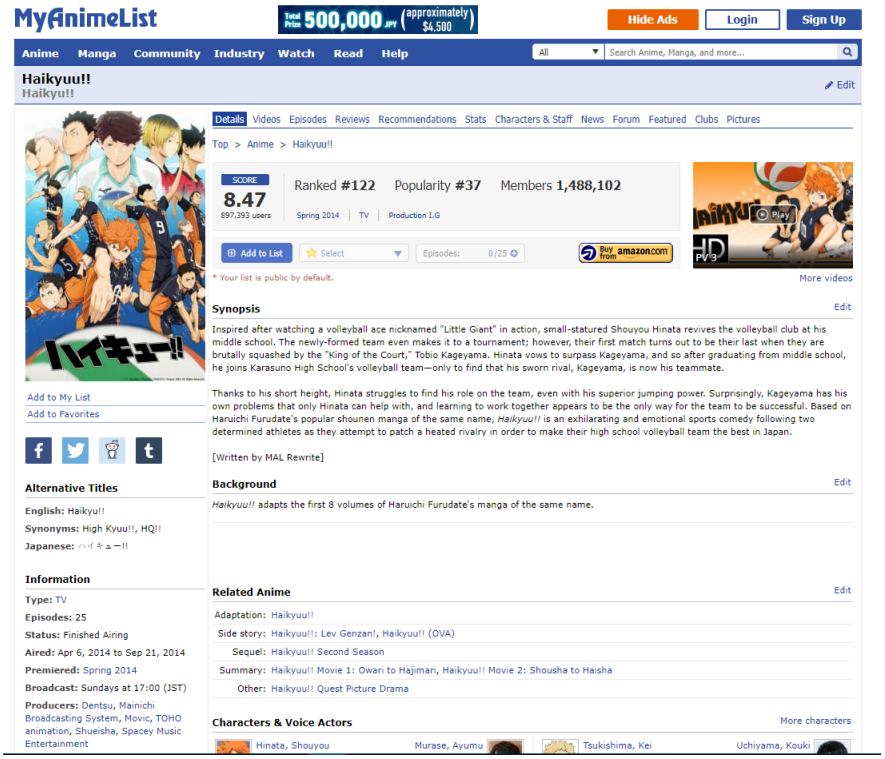

Figura 2.2: Detalles de un anime en MyAnimeList

2.3 RottenTomatoes

Rotten Tomatoes [3] es otra aplicación parecida a IMDb, con la diferencia de que las valoraciones las redactan críticos conocidos, evitando así el problema de que una seriei habitualmente sólo este valorada por *fanboys* o *haters*.

Aunque las críticas estén realizadas por profesionales, los usuarios pueden votar si una película les gusta o no. Finalmente, la valoración de la película o serie se calcula con el porcentaje de votos positivos frente al total de votos. Es decir, si 75 usuarios votaron que les gustaba y 25 votaron que no, la nota final es de un 75% (En la figura 2.3 se puede ver como se representan con porcentajes las notas).

Esta forma de voto acarrea un problema: se valora si la película le gustó o no al usuario, pero no cuánto le gusto. Es decir, el usuario se verá obligado a darle la misma nota a una película que le pareciese una obra maestra que a una que simplemente le gustó. Con esto los usuarios pueden comprobar cuánto es la aceptación de una película entre el público general, pero no cuánto les gustó.

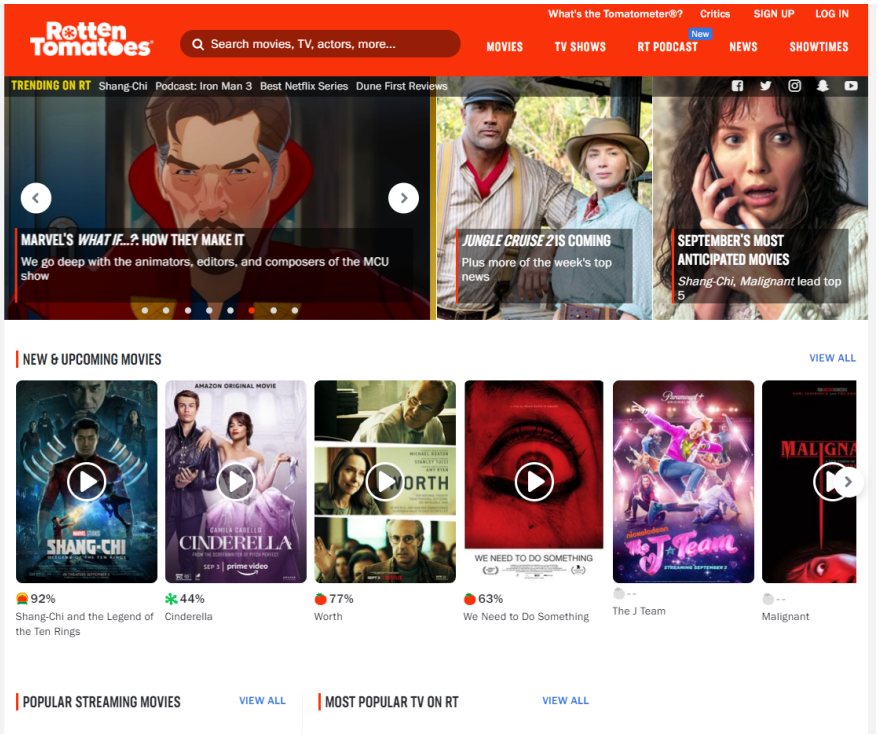

Figura 2.3: Página principal de Rotten Tomatoes

2.4 Metacritic

Metacritic [4] es otra página web destinada a puntuar series y películas, incluyendo juegos y álbumes de música.

Una característica de Metacritic es que usa un sistema diferente para los votos: Algunos tienen más peso que otros en la puntuación final, de forma que se intenta prevenir a los *fanboys* y *haters*.

Una ventaja de esta página es que muestra para cada ítem la nota y las críticas de los usuarios y de los críticos, pudiendo así compararlas como se muestra en la figura 2.4.

Una diferencia con la aplicación desarrollada, es que los usuarios publican críticas (que no comentarios), por lo que se puede llegar a perder un poco la interacción entre estos.

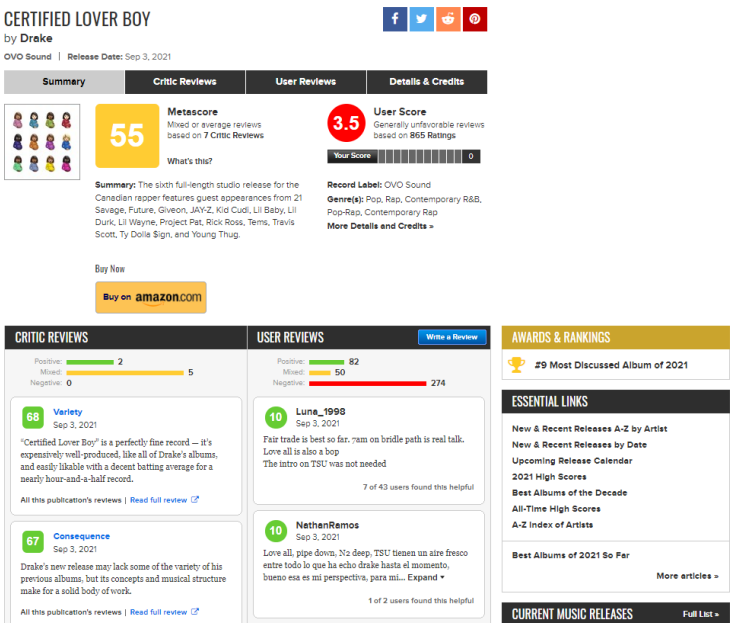

Figura 2.4: Detalles de un álbum en Metacritic

Capítulo 3

Metodología

La metodología es el marco de trabajo que sirve como guía para una correcta estructuración, planificación y construcción de un proyecto. Para este caso se optó por una metodología ágil debido a sus beneficios:

- Mejora la productividad de la calidad de los productos finales.
- Proporciona una gran respuesta a los cambios al ser un proceso evolutivo.
- Como se sigue un proceso iterativo e incremental, al final de cada etapa se obtiene un software funcional.
- Al estar pensada para adaptarse a los cambios, disminuye el riesgo del proyecto.

La metodología escogida es una metodología iterativa inspirada en *Scrum*[5], ya que no todas las características de este se aplican al proyecto.

3.1 Scrum

Scrum es una de las metodologías ágiles más utilizadas en el desarrollo software. Es un proceso en el que se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente en equipo y obtener el mejor resultado posible de un proyecto.

3.1.1 Roles

En *Scrum* se distinguen diferentes roles:

- **Product Owner:** Es el representante del cliente, cuya labor será la de optimizar y maximizar el valor del producto.
- **Scrum Master:** Es el responsable de asegurarse que el proyecto se reali siguiendo las bases de la metodología.

- **Equipo de desarrollo:** Cada uno de los profesionales que participan en la construcción del producto.

Dado que este proyecto se trata de un Trabajo de Fin de Grado, tanto el *Product Owner*, como el *Scrum Master* como el de equipo de desarrollo, son la misma persona. Por este motivo, los roles de *Scrum* no aplican a nuestra metodología.

3.1.2 Sprints

Los proyectos que siguen metodologías ágiles se dividen en iteraciones llamadas *sprints* los cuales abordan un subconjunto de las funcionalidades del sistema. Al final de cada *sprint* se debe obtener un producto que, potencialmente, se pueda entregar al cliente.

En cada uno de los *sprints* se realiza el análisis, diseño, implementación y pruebas de las tareas marcadas para realizar en la iteración.

3.1.3 Artefactos

Historias de usuario

Dado que un componente clave del desarrollo de software ágil es poner a las personas en primer lugar, las historias de usuarios ponen a los usuarios finales reales en el centro de la conversión. Las historias de usuario deberían ser independientes entre sí y deben tener un valor para el usuario, ser estimables y verificables y deben de tener una duración corta de forma que se puedan asumir varias historias de usuario en un mismo *sprint*.

Son la toma de requisitos básica de las metodologías ágiles y deben expresarse desde el punto de vista del cliente, de forma breve y con lenguaje informal:

*Como **[rol de usuario]** quiero **[acción]** para **[valor deseado]***

Product Backlog

El *Product Backlog* es una lista con todos los requisitos iniciales desde el punto de vista del cliente. Se trata de un lista dinámica, que evolucionará a medida que lo hace el producto. Estos requisitos se describen en forma de historias de usuario, con un lenguaje no técnico y priorizadas en base a su valor de negocio.

Sprint Backlog

El *Spring Backlog* es una lista de tareas del *Product Backlog* las cuales se llevarán a cabo durante un *sprint*. Esta lista la elabora el equipo en la reunión de planificación del *sprint*. Estos requisitos estarán ordenados por orden de prioridad para el cliente y deben ser lo suficientemente pequeñas para que se pueda detectar progreso o estancamiento de manera diaria.

Al finalizar cada *sprint* se dedica un tiempo a probar las nuevas funcionalidades implementadas en éste.

3.2 Aplicación de la metodología

La organización de artefactos en la práctica se realizó a través de la herramienta *Trello* (sección 6.3.2). Para cada *sprint* se generaba un tablero nuevo con las historias de usuario a acometer en él. Estas historias se organizaban en diferentes listas:

- *To-Do:* Se incluyen al principio todas las historias de usuario a realizar en el *sprint*.
- *WIP (Work In Progress):* Al comenzar con el diseño de una historia de usuario, se moverá de la pila de *To-Dos* a esta.
- *To-Test:* Una vez acabada el diseño y la implementación, se moverá la historia de usuario a esta lista para señalizar que la característica debe ser probada.
- *Bugged:* En el caso de encontrar algún error en el proceso de pruebas, se moverá la historia de usuario a esta lista y se agregará un pequeño mensaje que indique el error y como reproducirlo para su posterior solución.
- *Done:* Una vez acabadas satisfactoriamente las pruebas de una historia de usuario, se moverá a la pila de completadas .

La figura 3.1 muestra un ejemplo de como se trabajó con esta herramienta durante el proyecto.

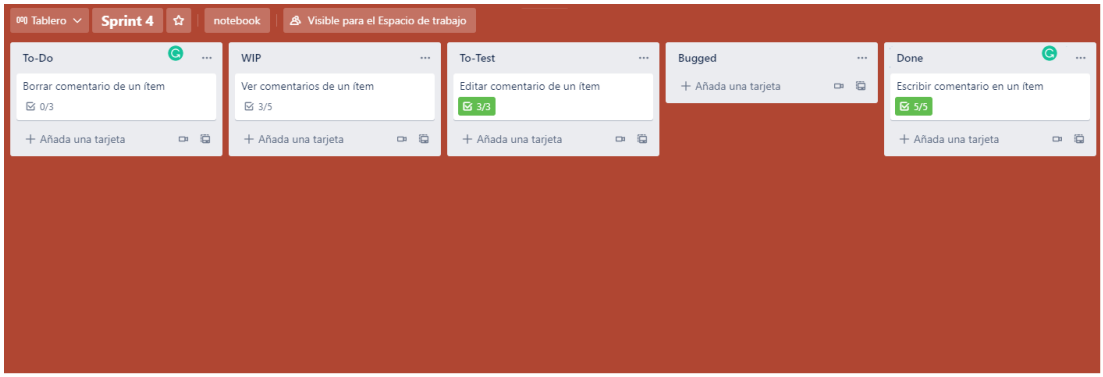

Figura 3.1: Tablero del Sprint 4

Capítulo 4

Análisis de requisitos global

4.1 Roles

En la aplicación hay dos tipos de actores: los usuarios autenticados y sin autenticar.

- **Usuario sin autenticar:** Los usuarios que no inicien sesión podrán acceder al catálogo de películas y series y ver su puntuación global y los comentarios que los distintos usuarios hayan realizado. También podrán ver los perfiles de los usuarios.

- **Usuario autenticado:** Los usuarios que inicien sesión, podrán puntuar y comentar las películas, puntuar los comentarios y seguir/dejar de seguir a los usuarios. Además, cada usuario podrá marcar las películas y series como ”Para ver”, ”Viendo” y ”Vista” para llevar un seguimiento.

4.2 Historias de usuario

Para la correcta lectura de este apartado, se definirá lo que en el proyecto se considera un *ítem*. Un ítem es una serie o película del catálogo.

En la tabla 4.1 muestra el Product Backlog con las historias de usuario.

ID	Nombre	Descripción
US01	Autenticación	Como usuario no autenticado quiero iniciar sesión en el sistema
US02	Registro	Como usuario no autenticado quiero registrarme en el sistema
US03	Cerrar sesión	Como usuario autenticado quiero cerrar sesión en el sistema
US04	Actualizar perfil	Como usuario autenticado quiero actualizar mi perfil
US05	Ver los ítems mejor puntuados	Como usuario quiero ver cuales son los ítems mejor puntuados
US06	Ver los ítems añadidos en el último mes mejor puntuados	Como usuario quiero ver cuales son los ítems añadidos en el último mes mejor puntuados mes
US07	Búsqueda de ítems	Como usuario quiero buscar ítems por el título
US08	Ver detalles de un ítem	Como usuario quiero ver los detalles de un ítem
US09	Escribir comentario en un ítem	Como usuario autenticado quiero escribir un comentario en un ítem
US10	Puntuar un ítem	Como usuario autenticado quiero puntuar un ítem
US11	Ver comentarios de un ítem	Como usuario quiero ver los comentarios de un ítem
US12	Puntuar comentario de un ítem	Como usuario autenticado quiero puntuar un comentario en un ítem
US13	Editar comentario de un ítem	Como usuario autenticado quiero poder editar mis comentarios realizados en un ítem
US14	Borrar comentario de un ítem	Como usuario autenticado quiero poder borrar mis comentarios realizados en un ítem
US15	Marcar ítems para seguir	Como usuario autenticado quiero poder llevar un seguimiento de ítems que tengo para ver, las que estoy viendo y las vistas
US16	Ver detalles usuario	Como usuario quiero ver los detalles de otros usuarios
US17	Seguir usuario	Como usuario autenticado quiero poder seguir a otros usuarios
US18	Dejar de seguir usuario	Como usuario autenticado quiero poder dejar de seguir a otros usuarios
US19	Ver usuarios seguidos	Como usuario quiero poder ver a los usuarios seguidos
US20	Ver comentarios realizados por un usuario	Como usuario quiero poder ver los comentarios de un usuario en concreto
US21	Ver ítems seguidos por el usuario	Como usuario autenticado quiero poder ver los ítems que tengo en seguimiento

Figura 4.1: Product Backlog

4.3 Funcionalidades

A continuación se describirá cada historia de usuario y se acompañará con algún Mockup que muestre la idea inicial del diseño.

US01 - Autenticación

El usuario no autenticado deberá proporcionar su nombre de usuario y su contraseña para iniciar sesión en el sistema. Si los datos son correctos, el usuario queda autenticado, si no, se mostrará un mensaje de error.

US02 - Registro

Para registrarse en el sistema, el usuario deberá proporcionar un nombre de usuario (único), su nombre, apellidos, contraseña y email como se muestra en la figura 4.2. Por seguridad, la contraseña se pedirá dos veces y se harán comprobaciones en que los datos cumplen el formato correcto (por ejemplo que el email contenga una "@"). Si los datos son correctos, se registra al nuevo usuario en el sistema y queda autenticado para la sesión.

Figura 4.2: Página de registro

US03 - Cerrar sesión

El usuario autenticado podrá cerrar sesión de la aplicación pulsando en el botón de salir, pasando a tener las funcionalidades de usuario no autenticado.

US04 - Actualizar perfil

Cualquier usuario autenticado, podrá modificar su perfil, lo que incluye su nombre, apellidos e email.

US05 - Ver los ítems mejor puntuados

Cualquier usuario podrá (en la pantalla principal de la aplicación), ver un carrusel con las películas y series con mejor valoración. El usuario verá una imagen, el nombre y la puntuación de esta, y podrá ir a sus detalles al clicar en la imagen.

US06 - Ver los ítems añadidos en el último mes mejor puntuados

Cualquier usuario podrá (en la pantalla principal de la aplicación), ver un carrusel con las películas y series con mejor valoración de las películas y series añadidas el último mes (figura 7.7). El usuario verá una imagen, el nombre y la puntuación de esta, y podrá ir a sus detalles al hacer clic en la imagen.

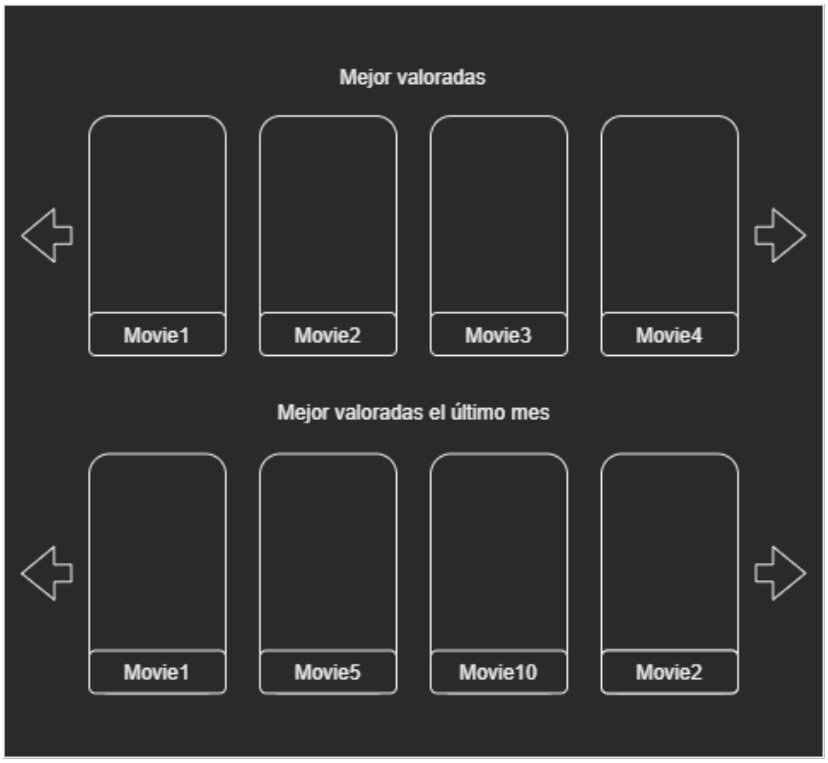

Figura 4.3: Página principal

US07 - Búsqueda de ítems

Cualquier usuario podrá buscar una película y serie por su título. Estas se mostrarán paginadas y ordenadas por la puntuación total y no en un carrusel. El usuario verá una imagen, el nombre y la puntuación de esta, y podrá ir a sus detalles al hacer click en la imagen.

US08 - Ver detalles de un ítem

Cualquier usuario podrá acceder a los detalles de una serie o película (figura 4.4). En sus detalles se mostrará la imagen, la puntuación media y la puntuación del usuario, la sinopsis, los géneros, la duración, su fecha de estreno y en el caso de ser una serie, sus temporadas o de ser una temporada, sus episodios. Cada elemento del catálogo, también incluirá (si lo tiene) un enlace a su página en los principales servicios de streaming, para poder acceder a él directamente.

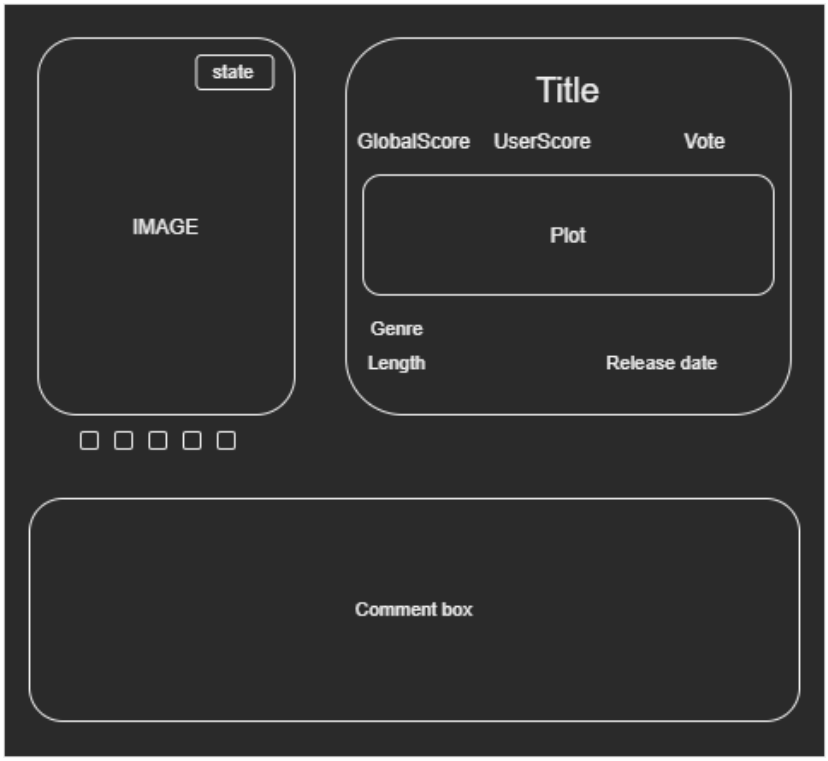

Figura 4.4: Página de detalles de un ítem

US09 - Escribir comentario en un ítem

Si el usuario está autenticado, podrá realizar comentarios en cada elemento del catálogo. El número de caracteres permitido es de 500 y además, un usuario podrá hacer todos los comentarios que quiera.

US10 - Puntuar un ítem

Los usuarios autenticados podrán puntuar una película o serie con una valoración de entre 0 y 10. Esta puntuación se mostrará al lado de la puntuación global. Si el usuario desea eliminar la puntuación, simplemente debe hacer clic encima de la puntuación que hizo.

US11 - Ver comentarios de un ítem

Cualquier usuario, al entrar en los detalles de un elemento del catálogo, debajo de sus detalles podrá ver los comentarios realizados (figure 4.5). Cada comentario tendrá el nombre del usuario que lo realizó (que será un enlace al perfil de ese usuario), el propio comentario, la fecha de creación o si se ha modificado, la de la última modificación y la puntuación total de ese comentario. Tambien incluye 3 botones: el de editar, el de puntuar positivamente y el de negativamente. Los comentarios vendrán paginados y ordenados por su fecha de creación de forma que se muestren primero los más antiguos.

Figura 4.5: Cajón de comentarios de un ítem

US12 - Puntuar comentario de un ítem

Los usuarios autenticados podrán puntuar cualquier comentario positivamente o negativamente. Para esto se hace uso de dos botones (pulgar arriba y pulgar abajo). Los botones en un principio aparecerán en blanco, pero cuando el usuario vote, se pintarán de azul o rojo dependiendo si votó positivo o negativo respectivamente. Al igual que con la puntuación de una película o serie, para retirar la puntuación vale con volver a pulsar la puntuación anterior.

US13 - Editar comentario de un ítem

Cualquier usuario registrado podrá editar sus propios comentarios pulsando el botón correspondiente (lápiz). Al clicarlo, los botones de los pulgares y el lápiz se cambian por el de una papelera, una aspa roja y un tick azul. Con el aspa roja se cancelará la edición del comentario y con el tick azul se guardará. La edición del comentario será en el propio comentario, el cual cambiará permitiendo la edición de este. Una vez editado, se modificará la fecha de creación por la de edición y aparecerá una marca que indica que el comentario fue editado.

Cabe destacar que un comentario no podrá ser modificado si previamente fue votado es decir, debe tener 0 votos positivos y negativos para poder ser modificado. Así el sistema se

asegura de que los usuarios no cambien el mensaje en función del número de votos.

US14 - Borrar comentario de un ítem

Cualquier usuario registrado podrá eliminar sus propios comentarios. Para ello debe clicar primero en el botón de eliminar y después en la papelera.

US15 - Marcar ítems para seguir

Los usuarios autenticados verán en los detalles de un ítem tres botones: un reloj, un ojo y un tick. Al pulsar estos botones, el usuario podrá cambiar el estado entre ”Para ver”, ”Viendo” y ”Vista”, respectivamente. Para desmarcar el estado de un ítem, llega con volver a pulsar la opción seleccionada.

US16 - Ver detalles usuario

Cualquier usuario (autenticado o no), podrá ver los detalles de un usuario (figura 4.6). Para cada usuario se mostrará su nombre, su apellido, su nombre de usuario, los usuarios a los que sigue y sus comentarios.

Si un usuario está viendo su propio perfil podrá además, ver las películas que tiene en seguimiento, el tiempo que ha pasado viendo películas y series y los géneros que más frecuenta.

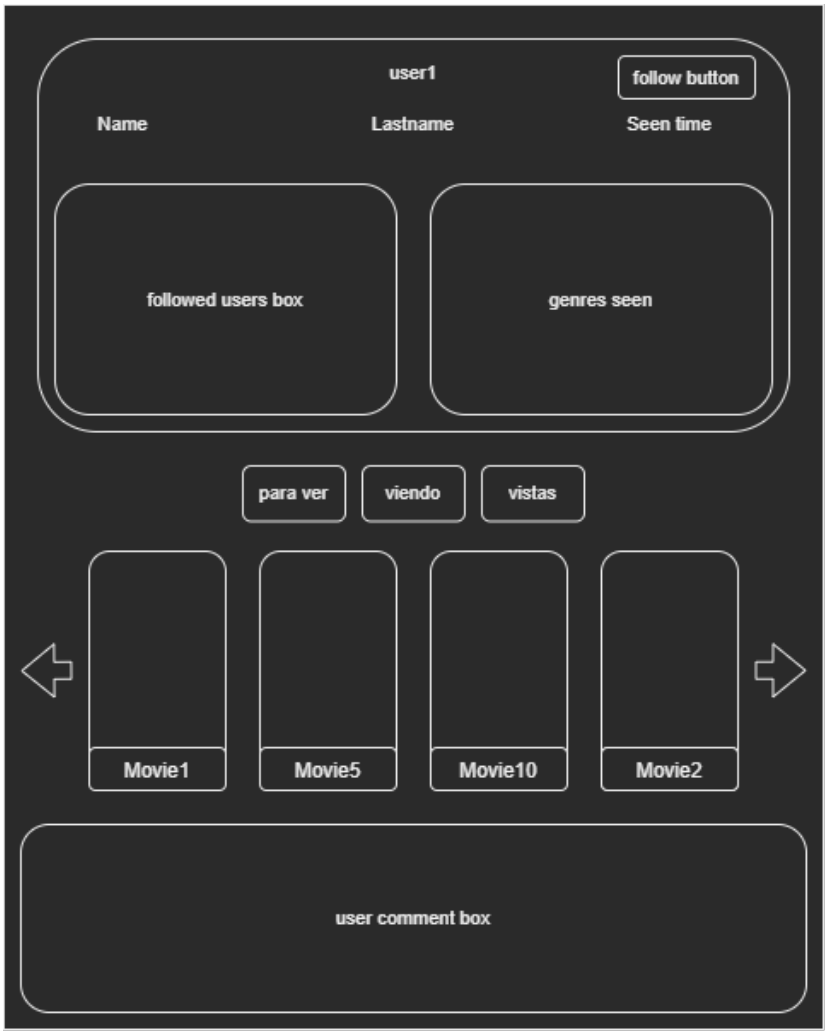

Figura 4.6: Página del perfil de un usuario

US17 - Seguir usuario

Un usuario autenticado que esté viendo el perfil de otro usuario al que no siga, tendrá un botón que le permitirá comenzar a seguir a ese usuario. En el caso de que ya siga a ese usuario, el botón cambiará por el de dejar de seguir.

US18 - Dejar de seguir usuario

Un usuario autenticado que esté viendo el perfil de otro usuario al que siga, tendrá un botón que le permitirá dejar de seguir a ese usuario. En el caso de que ya no ese usuario, el botón cambiará por el de empezar a seguir.

US19 - Ver usuarios seguidos

Cualquier usuario podrá ver los usuarios a los que sigue otro usuario. Se mostrará una lista con los usuarios a los que sigue paginados. Además, al hacer clic en el nombre de usuario nos llevará a su perfil.

US20 - Ver comentarios realizados por un usuario

Cualquier usuario podrá ver los comentarios realizados por un usuario. Se mostrará una lista con los comentarios realizados paginados y ordenados por fecha de creación. Además, al hacer clic en el comentario, nos llevará a los detalles de la película o serie en la cual se realizó el comentario.

US21 - Ver ítems seguidos por el usuario

Si un usuario autenticado está viendo su perfil, podrá ver las películas y series que tiene marcadas ”Para ver”, ”Viendo” y ”Vistas”.

Capítulo 5

Planificación

5.1 Sprints

Siguiendo las metodologías de desarollo iterativo, se han llevado a cabo distintos *sprints* en el proyecto, los cuales se han planificado de cara a repartir la carga de trabajo lo más equitativamente posible entre ellos.

5.1.1 Sprint 1: Concepción del producto

Este primer *sprint* se ha dedicado a estudiar las funcionalidades que debe tener el producto.

5.1.2 Sprint 2: Creación del proyecto base

En esta iteración se crea el proyecto base y se incoporan los siguientes casos de uso como arranque del proyecto:

- US01 - Autenticación
- US02 - Registro
- US03 - Cerrar sesión
- US04 - Actualizar perfil

5.1.3 Sprint 3: Búsqueda y visualización de ítems

En este *sprint* se implementan las primeras funcionalidades de la aplicación de cara a la búsqueda y visualización de series y películas. A este *sprint* corresponderían las siguientes historias de usuario:

- US05 - Ver los ítems mejor puntuados

- US06 - Ver los ítems añadidos en el último mes mejor puntuados
- US07 - Búsqueda de ítems
- US08 - Ver detalles de un ítem

5.1.4 Sprint 4: Gestión de comentarios

En esta iteración y ya con las funcionalidades de los ítems implementadas, se procede a desarrollar las historias de usuario referentes a los comentarios:

- US09 - Escribir comentario en un ítem
- US11 - Ver comentarios de un ítem
- US13 - Editar comentario de un ítem
- US14 - Borrar comentario de un ítem

5.1.5 Sprint 5: Gestión de puntuaciones

En este *sprint* se implementan las funcionalidades correspondientes a los votos, tanto de los ítems como de los comentarios:

- US10 - Puntuar un ítem
- US12 - Puntuar comentario de un ítem

5.1.6 Sprint 6: Seguimiento de los ítems

En esta iteración se desarrollan las funcionalidades correspondientes al seguimiento de las series y películas por parte de los usuarios, marcándolas como ”Para ver”, ”Viendo” y ”Vista”:

- US15 - Marcar ítems para seguir

5.1.7 Sprint 7: Interacción entre usuarios

En este *sprint* se implementan las funcionalidades relacionadas con el seguimiento entre usuarios y la visualización de los perfiles:

- US16 - Ver detalles de un usuario
- US17 - Seguir a un usuario
- US18 - Dejar de seguir a un usuario

- US19 - Ver usuarios seguidos
- US20 - Ver comentarios realizados por un usuario
- US21 - Ver ítems seguidos por el usuario

5.1.8 Sprint 8: Arreglo de bugs y mejoras

En este último sprint (a nivel de desarrollo de la aplicación) se arreglan diferentes errores que se observaron una vez la aplicación estaba acabada. También se implementaron diferentes mejoras en la aplicación..

5.1.9 Sprint 9: Elaboración de la memoria

El último *sprint* se dedica a la elaboración de la memoria.

5.2 Planificación temporal

Aunque en una planificación inicial se marcó una duración aproximada para cada *sprint*, durante varios momentos del proyecto no se ha podido dedicar el tiempo suficiente al proyecto debido a que se tuvo que compaginar la realización del TFG con prácticas extracurriculares. Por este motivo, algunos de los *sprints* se alargaron más de lo debido. En la siguiente tabla se muestra la duración estimada y la real de cada *sprint* (Cuadro 5.1).

Sprint	Objetivo	Fecha Inicio	Fecha Fin	Horas est.	Horas
1	Concepción del producto	01/04/21	20/04/21	12	15
2	Creación del proyecto base	21/04/21	13/05/21	20	29
3	Búsqueda y visualización de ítems	14/05/21	09/07/21	35	40
4	Gestión de comentarios	10/07/21	25/07/21	35	20
5	Gestión de puntuaciones	26/07/21	02/08/21	30	40
6	Seguimiento de los ítems	15/07/21	03/08/21	15	12
7	Interacción entre usuarios	04/08/21	09/08/21	30	41
8	Arreglo de bugs y mejoras	09/08/21	24/08/21	50	45
9	Elaboración de la memoria	24/08/21	10/09/21	80	90

Cuadro 5.1: Planificación temporal y horas dedicadas al proyecto

5.3 Cálculo de costes

Teniendo en cuenta las horas invertidas en la realización del proyecto (Cuadro 5.1) y tomando como coste medio 30€/hora, el coste estimado del proyecto sería de:

Coste del proyecto = 332h * 30€/hora = **9.960€**

Capítulo 6

Fundamentos tecnológicos

En el siguiente capítulo se explicarán las tecnologías y herramientas usadas durante el desarrollo de la aplicación:

- Tecnologías y herramientas empleadas en el desarrollo del *backend*
- Tecnologías y herramiendas empleadas en el desarrollo del *frontend*
- Tecnologías y herramientas complementarias

6.1 Tecnologías y herramientas empleadas en el desarrollo del backend

El *backend* de nuestra aplicación es una API REST implementada con Java EE y para gestionar la persistencia se usará una base de datos de MySQL.

6.1.1 MySQL

MySQL[6] es una base de datos relacional, considerada la base de datos de código abierto más popular del mundo. Aunque es de código abierto, MySQL es patrocinado por una empresa privada que posibilita su doble licenciamiento. Por un lado está la versión Community, distribuida bajo licencia pública, y por otra, la versión Enterprise. Algunas de las principales ventajas de MySQL son:

- Al ser una BBDD relacional, archiva los datos en tablas separadas, lo que permite tener mayor velocidad al realizar operaciones.

- Baja probabilidad de corrupción de datos.
- Al ser de código abierto, permite su modificación ajustándolo a las necesidades del usuario.
- Compatibilidad con SQL (Uno de los lenguajes más usados en la industria).

6.1.2 Java

Java[7][8] es uno de los lenguajes de programación más usados en el mundo. Dos de sus características es que se basa en el paradigma de la programación orientada a objetos y que permite su ejecución en múltiples sistemas operativos.

Al ser orientado a objetos, permite que el código se reutilice, tanto en el mismo proyecto como en proyectos diferentes. Además, Java compila el código fuente en Java bytecode, permitiendo que se ejecute en cualquier máquina donde haya una Java Virtual Machine (JVM).

6.1.3 Maven

Maven[9] es una herramienta que permite la gestión y construcción de proyectos Java. Utiliza el POM que es un fichero XML en el que se describe el proyecto a construir, sus dependencias y el orden de construcción de los elementos. Algunas de sus ventajas son:

- Es un sistema de gestión de dependencias.
- Permite la creación de plugins customizables.
- Al ser una aplicación Java, funciona en cualquier sistema operativo siempre que haya una JVM.
- Es software libre, lo que permite la modificación y customización del código en caso de ser necesario.
- Fomenta la reutilización de código y de librerías.

6.1.4 Spring Boot

Spring[10] es un framework para el desarrollo de aplicaciones Java. Su objetivo es simplificar y facilitar la construcción de aplicaciones Java EE. Algunas de sus ventajas son:

- Inyección de dependencia: Patrón de diseño utilizado como una de las formas de inversión de control favoreciendo el bajo acoplamiento. Cuando una clase necesita a otra, Spring se lo inyecta, siguiendo el principio de Hollywood, en el cual es la aplicación la que tiene el control y no el código. Esto nos permite una mayor modularidad, escalabilidad y evita la dependencia entre clases.

- Minimiza el código repetitivo.

- Simplifica el acceso a datos.

Spring Boot[11] por su parte, simplifica el proceso de creación y configuración de la aplicación gracias a:

- **Contenedor de aplicaciones integrado:** Spring Boot integra el servidor de aplicaciones en el propio .jar y lo levanta cada vez que se arranca la aplicación. Esto permite una mayor facilidad en la distribución de aplicaciones.

- ***Starters:*** Dependiendo de lo que se necesite, Spring Boot nos facilita una serie de dependencias, llamadas *starters* que se pueden añadir al proyecto, proporcionando todas las dependencias necesarias. Estos *starters* ya vienen con valores por defecto, lo que minimiza el trabajo de configuración.

6.1.5 REST

REST (REpresentational State Transfer) es un estilo de arquitectura software para sistemas de la web. Está diseñada para funcionar con sistemas distribuidos centralizados y con arquitecturas cliente-servidor, haciendo que sea el cliente el que guarde y mantenga el estado.

6.1.6 IntelliJ IDEA

IntelliJ IDEA[12] es un IDE desarrollado por JetBrains[13] que cuenta con dos ediciones: la Community y la Ultimate. Está disponible para todos los sistemas operativos y soporta infinidad de lenguages de programación. Una característica importante de IDEA es el amplio mercado de plugins que tiene, con casi 3000 plugins entre las dos versiones[14].

6.2 Tecnologías y herramiendas empleadas en el desarrollo del frontend

En esta sección se explicarán las tecnologías y herramientas usadas en el desarrollo del *frontend* de la aplicación.

6.2.1 JavaScript

JavaScript (JS) es un lenguaje de programación interpretado, orientado a objetos, débilmente tipado y dinámico. Actualmente implementa la versión ECMASCript2020[15], 11ª versión del estándar ECMAScript. En esta versión se incluye el tipo primitivo BigInt, los operadores de unión nula y el encadenamiento opcional, que permite el acceso a propiedades anidadas de objetos sin tener que verificarlo.

Aunque también se puede usar JS en el *backend*, lo habitual es que se use en el *frontend*, ya que es el idioma de programación más usado en la web.

Actualmente la mejor forma de trabajar con JS es haciendo uso de los frameworks que se han desarrollado para ello. Algunos de los más importantes ordenados por uso son:

1. React[16]
2. Angular[17]
3. Vue.js[18]
4. Svelte[19]
5. Preact[20]
6. Ember[21]

6.2.2 React

React[16] es actualmente el framework de JS más usado (casi un 80% de los programadores lo usó en 2020 y con una satisfacción del 88%). Su objetivo es facilitar la creación de interfaces de usuario (sería la Vista en el patron Modelo-Vista-Controlador) y aplicaciones SPA. Está desarrollado por Facebook[22] y la comunidad de software libre. Actualmente nuestra aplicación usa la versión 16.14.0, aunque la versión más reciente es la 17.0.2.

Virtual DOM

Una característica importante de React es su *Virtual DOM* (Document Object Model), el cual después se compara con el Document Object Model (DOM) del navegador para saber que partes han cambiado y actualizarlo eficientemente.

Componentes

React está orientado a la reutilización de componentes, permitiéndonos modularizar el código ahorrándonos así la escritura de código innecesario. Además, cada componente tiene sus *props*, que vendrían siendo sus atributos y pueden ser *statefull* o *stateless* dependiendo de si tienen estado o no.

JSX

Una extensión importante de JS es JSX, la cual nos facilita la creación de interfaces de usuario al asemejarse a como sería en HTML, aunque su uso no es obligatorio, es recomendable.

```
1  class Component extends React.Component {
2      render(){
3          return(
4              <div>
5                  <h1>Hola mundo</h1>
6              </div>
7          )
8      }
9  }
```

Figura 6.1: Código con JSX

Life Cycles

Otra característica importante de React son los ciclos de vida por los cuales pasan los componentes *statefull*:

- Con *shouldComponentUpdate* se puede prevenir el re-renderizado innecesario de un componente.
- *ComponentDidMount* se suele usar cuando el componente se ha creado en la interfaz.
- *componentWillUnmount* se suele llamar justo antes de que el componente se desmonte, siendo usado para limpiar las dependencias del componente.
- *render* se llama cada vez que el estado del componente es actualizado.

6.2.3 Redux

Redux[23] es un contenedor predecible del estado de aplicaciones de JS. Esta librería de código abierto está destinada al manejo del estado de las aplicaciones y usualmente se usa con React o Angular, y está inspirada en Flux[24], una librería de Facebook.

El funcionamiento de Redux es simple:

- El estado de la aplicación se guarda en un objeto el cual sólo puede ser modificado a través de acciones, para que no se puede cambiar el estado arbitrariamente.
- Estas acciones tambien son objetos en JS que describen lo que ocurre, facilitándonos el rastreo de errores.
- Finalmente se usa un reductor para juntar el estado y las acciones entre sí. El *reducer* toma el estado de la aplicación y la acción a llevar a cabo y devuelve el siguiente estado de la aplicación.

Redux tiene tres principios básicos:

- **Única fuente de la verdad:** el estado de toda la aplicación está almacenado en un árbol guardado en un único *store*.
- **El estado es de solo lectura:** La única forma de modificar el estado es emitiendo una acción, el cual debe devolver un nuevo estado.
- **Los cambios se realizan con funciones puras:** Para especificar como el árbol de estado es transformado por las acciones, se utilizan *reducers* puros.

6.2.4 React+Redux

Cuando se usa Redux con un framework para interfaces de usuario, es necesario usar una librería para vincular Redux con ese framework. Sin embargo, como React y Redux son comunmente usadas juntas, el equipo oficial de Redux desarrolló *React Redux*[25], que es la librería de *binding* oficial para React.

6.2.5 npm

npm[26] o *Node Package Manager*, como su nombre indica, es el sistema de gestión de paquetes por defecto para Node.js[27]. Su funcionamiento es sencillo, npm revisa las dependencias del proyecto en el fichero *package.json*, en el que se marcan para cada dependencia el rango válido de versiones.

6.2.6 MaterialUI

MaterialUI[28] es un framework de CSS y JavaScript que proporciona componentes de React para un desarrollo rápido y sencillo de interfaces de usuario. Aunque es gratuito, también proporciona temas premium, aunque en el proyecto se usa unicamente la versión gratuita.

6.2.7 Visual Studio Code

VSCode[29] es un editor de código gratuito creado por Microsoft que cuenta con una infinidad de plugins para ajustarlo a las necesidades del desarrollador. Algunas de sus ventajas son:

- Su *IntelliSense*, que ayuda a producir código mucho más rápido gracias a su autocompletado.
- Su compatibilidad con *Git*.
- Su extenso mercado de plugins[30].
- Que es multiplataforma.
- Su ligereza.

6.3 Tecnologías y herramientas complementarias

6.3.1 Git

Git[31] es el software de control de versiones SCM más usado actualmente. Una de sus principales diferencias frente a otros SCM es que es un sistema distribuido en el que cada usuario guarda localmente una copia del histórico de las versiones

6.3.2 Trello

Trello[32] es una herramienta web destinada a la organización de actividades. Se basa en el modelo *Kanban* en el cual las actividades se añaden a un tablero como tarjetas las cuales se pueden ir desplazando entre las diferentes fases por las que pueda pasar la tarea (Por ejemplo: *To do, In progress, Done*)

6.3.3 Postman

Postman[33] es una *API platform* desarrollada para construir y usar APIs. Su interfaz gráfica es muy sencilla de comprender y su facilidad de uso permite probar nuestra API REST de una forma muy rápida y cómoda.

6.3.4 DataGrip

DataGrip[34] es un IDE orientado a bases de datos desarrollado por JetBrains (al igual que IntelliJ IDEA). Aunque IntelliJ IDEA ya proporciona herramientas para trabajar con BBDD, DataGrip es mucho más completo y su interfaz de usuario está mucho más adaptada que la Intellij IDEA.

Capítulo 7

Desarrollo iterativo

En este apartado se detallarán los aspectos más importantes del desarrollo de la aplicación.

7.1 Modelo de datos

La aplicación (como se puede ver en la figura 7.1) gira entorno a la entidad *CatalogItem*, de la cual heredarán *Movie, Serie, Season* y *episode* que representan las películas, series, temporadas de una serie y los episodios de una temporada respectivamente. Entre estas entidades, los episodios estarán relacionados con una temporada y las temporadas con una serie.

Estos ítems podrán tener uno o más géneros, representados con la entidad *Genre* y también habrá enlaces que dirijan directamente a las páginas para ver esos ítems en los servicios de streaming correspondientes, representados por la entidad *StreamingCompany*.

Los ítems podrán recibir puntuaciones y comentarios (entidades *Score* y *Comment* respectivamente). Ámbos serán realizados por los usuarios (entidad *User*) que podrán tambien puntuar los comentarios realizados (*CommentScore*).

Además habrá una entidad *LastUpdate* que se traduce en una tabla con una única fila y columna que indica la fecha en la que se hizo por última vez una consulta al API externa para obtener nuevos ítems. Más adelante, en el apartado 7.4.2, se explica como se realiza.

7.2 Sprint 1: Concepción del producto

Este *sprint* consistía en pensar que funcionalidades debe tener la aplicación. El objetivo principal es el de puntuar y permitir llevar un seguimiento de las películas y series de los principales servicios de streaming.

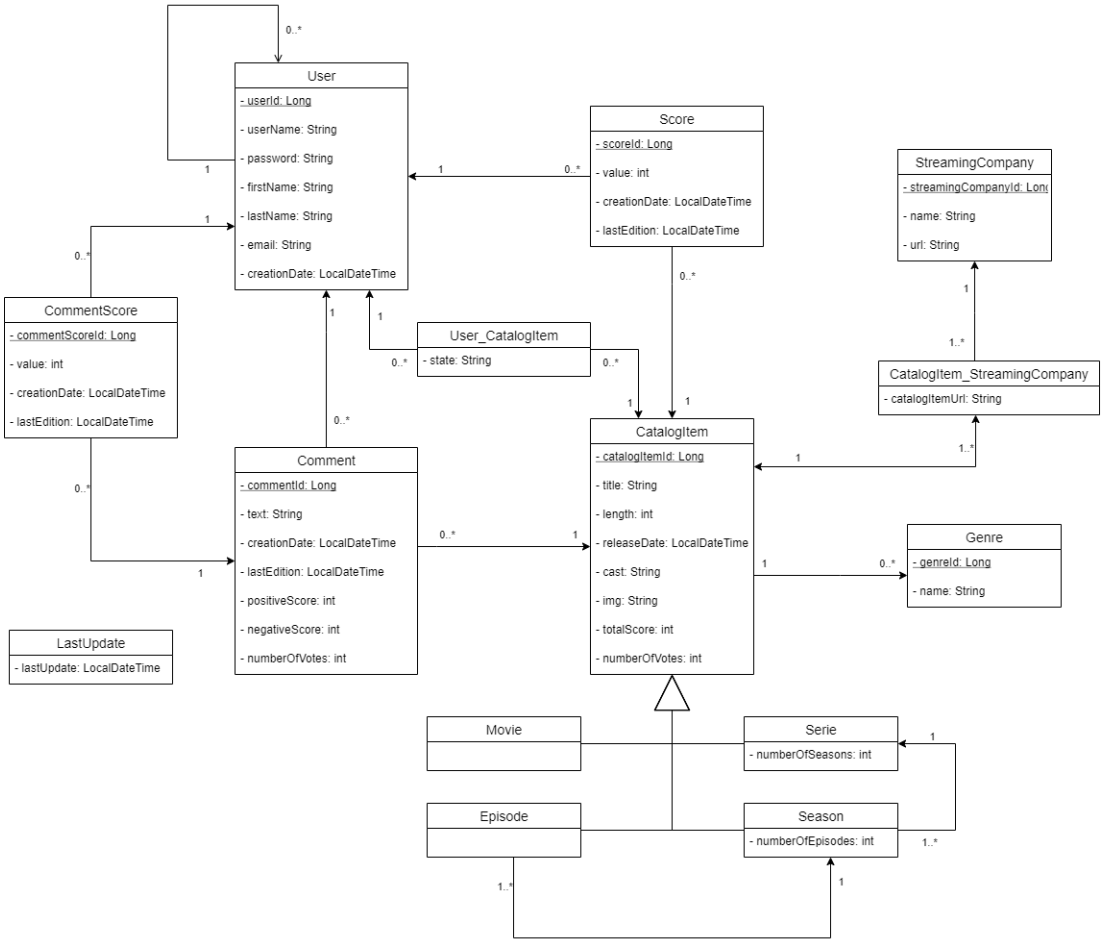

Figura 7.1: Diagrama de entidades de la aplicación

7.3 Sprint 2: Creación del proyecto base

En este *sprint* se llevó a cabo la configuración e instalación del entorno de desarrollo y de la base del proyecto.

7.3.1 Construcción de la estructura del *backend*

Para la creación del *backend* de la aplicación, se siguió la estructura base de un proyecto Maven:

- **src/main/java:** Contiene la estructura de paquetes con el código fuente de la aplicación. En la raíz se encuentra la clase *Application.java*, con las anotaciones *@SpringBootApplication* (para que la aplicación se despliegue y arranque automáticamente en un servidor Tomcat que tiene Spring Boot integrado) y *@EnableScheduling* para habilitar a Spring la ejecución de tareas programadas.
- **src/main/resources:** Contiene el archivo application.yml que es un fichero de propiedades del proyecto. También contiene los ficheros de internalización de la aplicación, en este caso en español, gallego e inglés.
- **src/sql:** Contiene los ficheros de creación de las tablas en base de datos y de algunos datos que se creen al inicio de la aplicación como por ejemplo las compañías de streaming y los géneros de los ítems.
- **src/test/java:** Contiene los paquetes de clases donde se implementan las pruebas de la aplicación.
- **src/test/resources:** Contiene un archivo application-test.yml con la configuración que requieren las pruebas automatizadas.

Para la persistencia de datos, se crearon dos instancias: una para la aplicación y otra para las pruebas. Una vez construido el proyecto base, se configuró un repositorio de Git en GitHub, tanto para poder trabajar desde otros dispositivos como backup en caso de haber algún problema durante el desarrollo de la aplicación.

7.3.2 Construcción de la estructura del *frontend*

Como ya se indica en el apartado 1.3, el *frontend* de la aplicación se desarrolla con los frameworks React y Redux, con lo que la estructura de ficheros quedaría de la siguiente forma (figura 7.2).

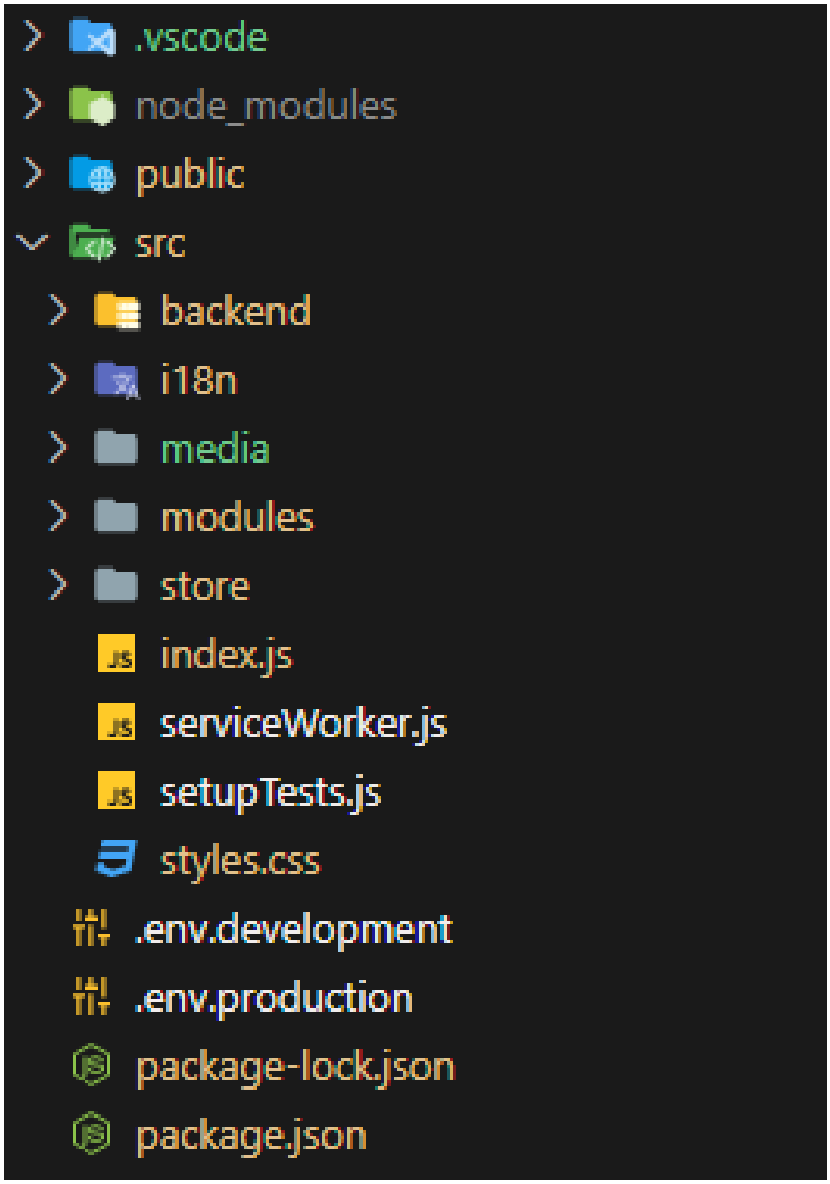

Figura 7.2: Estructura del frontend de la aplicación

Los directorios *package.json* y *package-lock.json* son ficheros de configuración que contienen la información referente a las dependencias del proyecto. El resto de directorio y ficheros salvo *src* son autogenerados con el proyecto. A continuación se describe el contenido del directorio *src* que contiene el código fuente del *frontend* de la aplicación:

- **src/backend:** En este directorio se encuentran los archivos necesarios para la conexión entre el *frontend* y el *backend* de la aplicación.
- **src/i18n:** Este directorio contiene todo lo referente a la internalización de la aplicación, que al igual que en el *backend*, se contemplan el castellano, el gallego y el inglés.
- **src/media:** En este directorio hay diferentes imágenes que se necesitan en la aplicación.
- **src/store:** Este directorio contiene los archivos en donde se crea el estado de Redux global de la aplicación.
- **src/index.js:** En este archivo se configura la *store* de la aplicación, el proxy del *backend*, la internalización de la aplicación y el renderizado de la aplicación.
- *styles.css:* En este fichero se implementan algunos estilos de la aplicación.
- **src/modules:** Este directorio contiene la mayor parte de código fuente del *frontend*. Como se puede observar en la figura 7.3, esta carpeta contiene un directorio *app* que contiene los ficheros de los componentes más globales de la aplicación como pueden ser *Body.js* o *Header.js*, los cuales como su nombre indica implementan el cuerpo y la cabecera de la aplicación. Contiene también un directorio por cada uno de los servicios que hay en el *backend*, en este caso hay dos servicios: el de *users* y el de *catalog*. También hay otro directorio, *common* en el cual se implementan componentes que reutilizarán a lo largo de la aplicación independientemente módulo.

  A continuación se explicará la jerarquía del módulo *catalog*, ya que la estructura es común a todos:

  - **catalog/components:** Este directorio contiene los diferentes componentes relacionados con el catálogo. Cada componente tiene tres ficheros: el fichero *styles.css* donde se implementan los estilos del componente, *componentName.js* donde se implementa la lógica del componente y un fichero *index.js* desde el cual se exportará el componente para su posterior uso.
  - **catalog/actions.js:** En este fichero se implementan las acciones de Redux del módulo correspondiente.
  - **catalog/actionTypes.js:** Este fichero contiene las constantes de los tipos de acciones correspondientes.

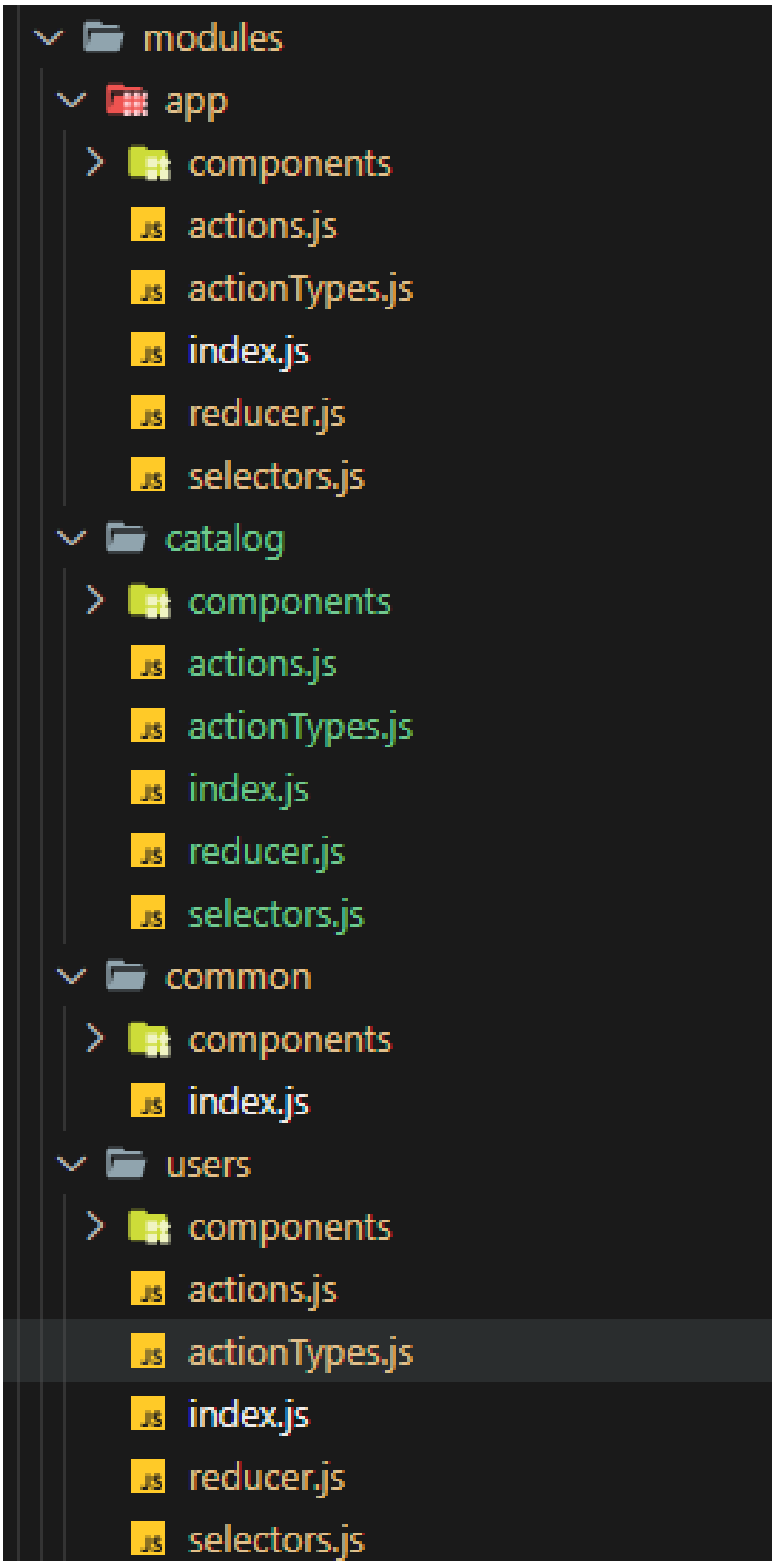

Figura 7.3: Estructura del directorio modules del frontend de la aplicación

- **catalog/index.js:** Este archivo exporta el resto de archivos del directorio.
- **catalog/reducer.js:** Este fichero implementa el *reducer* de Redux. Cada módulo tiene el suyo, los cuales después se combinan en el fichero *src/store/rootReducer.js*.
- **selectors.js:** En este fichero se implementan las funciones selectoras. Estas funciones reciben un estado de Redux como argumento y devuelven los datos derivados de ese estado.

7.3.3 Análisis

En esta iteración se implementaron las funcionalidades relacionadas con la gestión de usuarios. En las tablas 7.1, 7.2 y 7.3 se define el *Sprint Backlog* con las historias de usuario correspondientes.

US01 - Autenticación
Como usuario no autenticado quiero iniciar sesión en el sistema
Tareas
Implementación del servicio de autenticación en el *backend*
Creación de la vista de login
Implementación en el *frontend* de las acciones, estados y *reducer* de Redux necesarios para la autenticación

Cuadro 7.1: Sprint 2: Autenticación

US02 - Registro
Como usuario no autenticado quiero registrarme en el sistema
Tareas
Implementación del servicio de registro en el servicio REST
Creación de la vista de registro
Implementación de la validación de datos en el formulario de registro

Cuadro 7.2: Sprint 2: Registro

US03 - Cerrar sesión
Como usuario autenticado quiero cerrar sesión en el sistema
Tareas
Implementación del logout en el *fronent*

Cuadro 7.3: Sprint 2: Cerrar sesión

7.3.4 Diseño e implementación

Para la implementación de la autenticación se uso el estándar Jason Web Token (JWT),ya que se integra fácilmente con *Spring Security* y permite que el API sea completamente *stateless*, es decir, el *backend* no tiene que mantener un registro de los tokens, sino que es el cliente el que se encarga de almacenarlo y enviarlo en cada petición.

Cada token es autoconocido, compacto e incluye los datos necesarios para comprobar su validez e identificar correctamente al usuario que realiza la petición. En la figura 7.4 se puede observar el flujo de ejecución de una autenticacón con JWT.

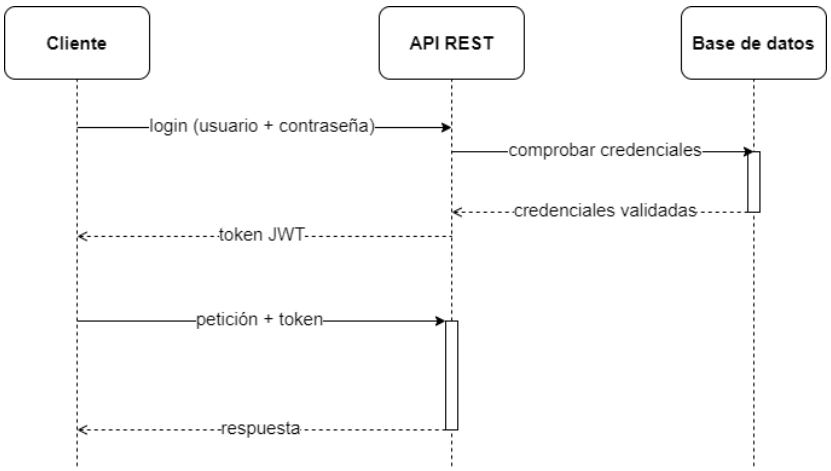

Figura 7.4: Autenticación don JWT

En la figura 7.5 se se muestran las clases que participan en el proceso de autenticación. *UserController* es el controlador REST que se encargará de recibir las peticiones relacionadas con la gestión de usuarios. *UserServiceImpl* es la implementación de *UserService* que implementará la lógica de negocio de los servicios relacionados con la gestión de usuarios. El servicio usará el *UserDao* para acceder a la base de datos y también usará *JwtGenerator* (implementada por *JwtGeneratorImpl*) para generar el JWT correspondiente al usuario. Esta clase también será usada por *SecurityConfig* (que extiende de *WebSecurityConfigAdapter*) que contiene la configuración de seguridad de Spring.

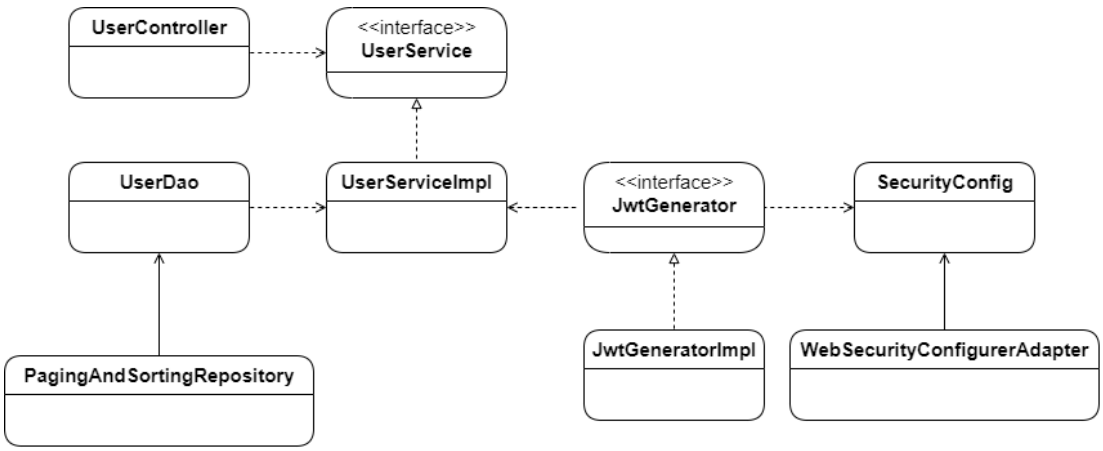

Figura 7.5: Diagrama de clases implicadas en el proceso de autenticación

En el *frontend* de la aplicación, en el módulo de *user* se creó el componente *Login*, el cual incluye un formulario para que le usuario introduzca el nombre de usuario y la contraseña. Para mantener la información del usuario, en el *store* de Redux *users* se agrega la propiedad *user* que guardará la información del usuario durante una vez inicie sesión.

Como al recargar la página, el estado de Redux se limpia, se debe guardar el JWT en el *Session Storage* del navegador, para que el usuario no tenga que autenticarse cada vez que recargue la página.

Para el registro de usuario en la aplicación se implementó una operación POST en el API REST que recibe los datos del usuario. Para esto, el usuario debe rellenar un formulario con todos los datos pertinentes. Primero los datos se validarán sintácticamente en el *frontend* de la aplicación (por ejemplo que el email contenga una "@"). Después en el *backend* se encargará de verificar que los datos son correctos y de serlo, los almacenará en la base de datos y generará un JWT que se enviará en la respuesta de la petición.

Para el cierre de sesión de un usuario en la aplicación, el *frontend* sencillamente eliminará el JWT del *Session Storage*. De esta forma en la próxima petición que realice no podrá enviar dicho token y la aplicación lo tratará como un usuario no autenticado.

7.4 Sprint 3: Búsqueda y visualización de ítems

7.4.1 Análisis

En esta iteración se implementan las historias de usuarios referentes a la búsqueda y visualización de los ítems. En las tablas 7.4, 7.5, 7.6, 7.7 se pueden ver las historias de usuario

que se llevaron a cabo en el *sprint*.

US05 - Ver los ítems mejor puntuados
Como usuario quiero ver cuales son los ítems mejor puntuados
Tareas
Implementación del servicio de obtención de ítems
Implementación de las peticiones en el controlador correspondiente
Diseño del *frontend* de la visualización de los ítems
Implementación del diseño del frontend

Cuadro 7.4: Sprint 3: Ver los ítems mejor puntuados

US06 - Ver los ítems añadidos en el último mes mejor puntuados
Como usuario quiero ver cuales son los ítems añadidos en el último mes mejor puntuados mes
Tareas
Implementación del servicio de obtención de ítems añadidos en el último mes
Implementación de las peticiones en el controlador correspondiente
Diseño del *frontend* de la visualización de los ítems
Implementación del diseño del frontend

Cuadro 7.5: Sprint 3: Ver los ítems añadidos en el último mes mejor puntuados

US07 - Búsqueda de ítems
Como usuario quiero buscar ítems por el título
Tareas
Implementación del servicio de búsqueda de ítems
Implementación de las peticiones en el controlador correspondiente
Diseño del *frontend* de la visualización de los ítems
Implementación del disñ

Cuadro 7.6: Sprint 3: Búsqueda de ítems

US08 - Ver detalles de un ítem
Como usuario quiero ver los detalles de un ítem
Tareas
Implementación del servicio de obtención de los detalles de un ítem
Implementación de las peticiones en el controlador correspondiente
Diseño del *frontend* de la página de detalles de un ítem
Implementación del disñ

Cuadro 7.7: Sprint 3: Ver detalles de un ítem

7.4.2 Diseño e implementación

En este sprint se creó toda la estructura necesaria para poder obtener ítems, buscarlos y ver sus detalles. Como se ve en la figura 7.6, *CatalogController* es el controlador que se encarga de recibir las peticiones sobre los ítems del catálogo. La lógica de negocio se encuentra en la clase *CatalogServiceImpl* (que implementa la interfaz *CatalogService*). A través de *CatalogItemDao*, *GenreDao* y *StreamingCompanyDao* se accede a la base de datos. Tambien se usa la clase *UtilsImpl* (que implementa la interfaz *Utils*) que contiene métodos que se usan tanto en el *CatalogService* como en el *UserService*.

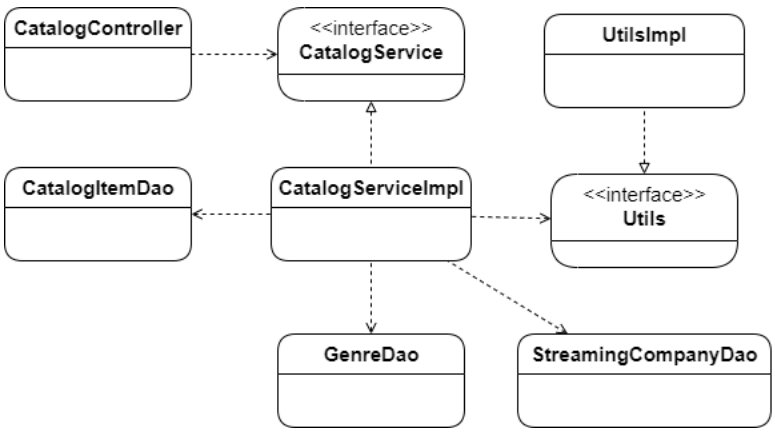

Figura 7.6: Diagrama de clases implicadas en los procesos de las operaciones sobre ítems

En cuanto al *frontend* de la aplicación, una de las mayores dificultades fue diseñar una interfaz de usuario bonita y sencilla de usar (figura 7.7).

Para la correcta visualización de los ítems en la página principal de la aplicación se decidió incorporar carruseles para mostrar el mayor número de ítems en el menor espacio posible, ya que se buscó que la página principal no precisara una barra de scroll para verlo en su totalidad.

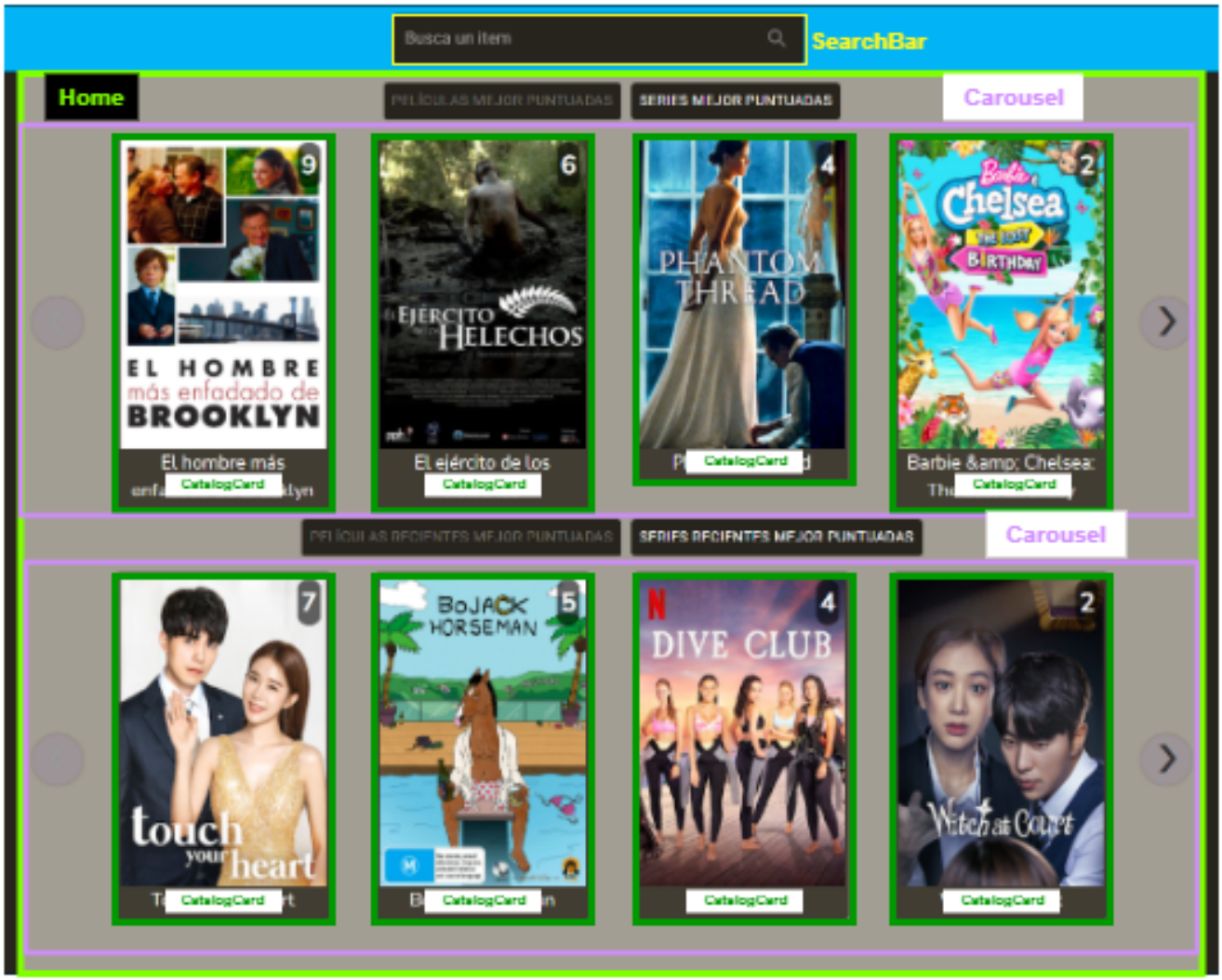

Figura 7.7: Componente *Home*

El carrusel que se implementa en la aplicación es un componente de la librería *react-elastic-carousel*[35] el cual es altamente modificable de cara a ajustarlo a las necesidades del proyecto. Este componente recibe un array de *CatalogCard*, un componente implementado en la aplicación. Este componente muestra la imagen del ítem correspondiente, su nombre y su puntuación de una forma compacta (como se puede ver en la figura 7.8).

Figura 7.8: Componente *CatalogCard*

Por defecto la aplicación muestra dos carruseles: el primero con las películas mejor valorades y el segundo con las películas añadidas el último mes mejor valoradas. Para cambiar a las series, basta con hacer clic en el botón correspondiente. Estos botones tienen un estado interno el cual les indica si deben estar habilitados o deshabilitados. Al hacer clic en uno de los botones, se hace una petición al *backend*, el cual devolverá los datos correspondientes y se almacenarán en el estado de Redux.

La búsqueda de ítems se hará a través del componente *SearchBar* enviando un texto a buscar. El *backend* devolverá todos los artículos del catálogo que contengan en su título el texto a buscar ordenados por su puntuación media y de manera paginada. Si no se escribe nada, el *backend* devuelvolverá todos los ítems del catálogo (también de forma paginada).

Al igual que en los carruseles, en la búsqueda de ítems la aplicación mostrará una lista de

CatalogCard. Al hacer clic en un ítem, se redigirá al usuario a la página de detalles del ítem a la vez que se envía una petición al *backend* buscando obtener todos los detalles de un ítem.

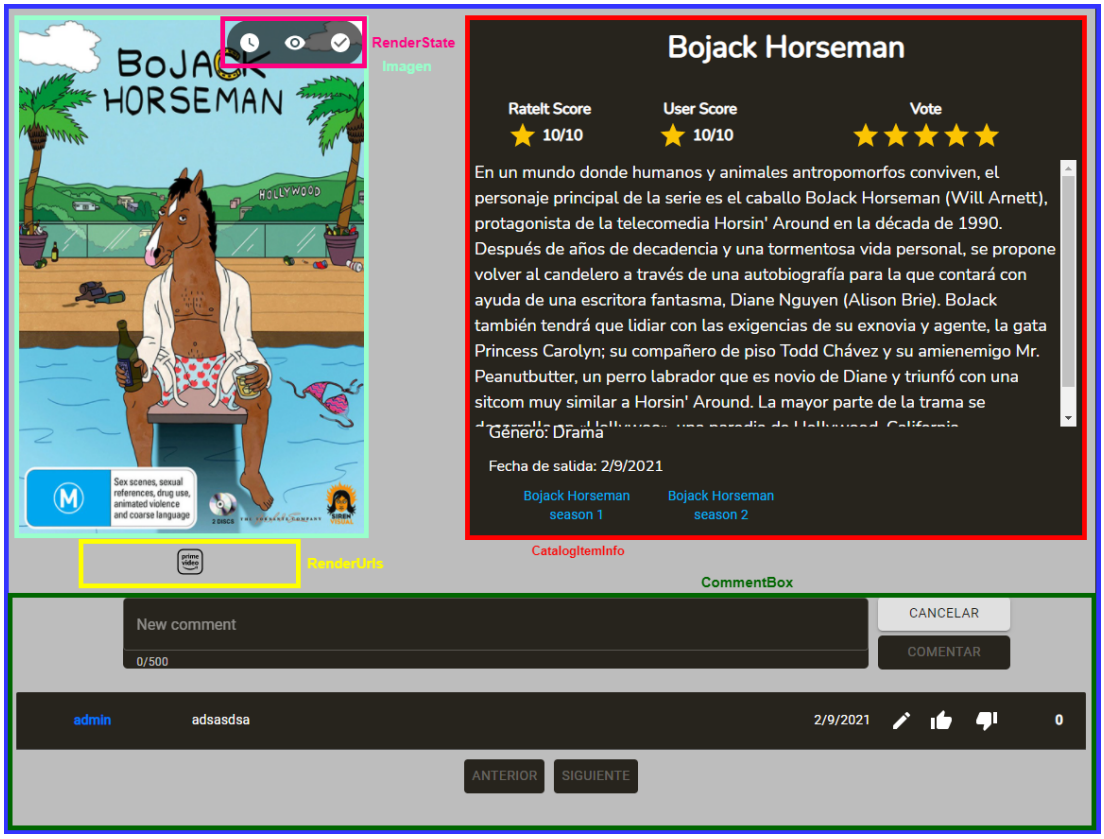

Figura 7.9: Detalles de un ítem del catálogo

Como se puede ver en la figura 7.9, la vista de los detalles de un ítem está dividida en componentes:

- **RenderState:** Este componente se encarga de renderizar los botones que controlarán el seguimiento de los ítems por parte los usuarios.
- **RenderUrls:** La función de este componente es el de renderizar los botones que redirijan a las urls concretas de cada ítem en cada uno de los servicios de streaming (siempre y cuando se proporcionen).
- **CatalogItemInfo:** Este componente se encarga de mostrar todos los datos que haya de cada ítem. También incluye el componente que se encarga de gestionar el voto de un ítem.
- **CommentBox:** Este componente incluye la caja de comentarios del ítem. Se hablará

más adelante en profundidad de él.

En esta iteración también se implementa la función de obtener los datos del API externa y guardarlos en la base de datos. Para ello se implementa el método *schedule()* que tiene la anotación *@Scheduled(fixedRate = 86400000)*. Con esta anotación el método queda programado para ejecutarse periódicamente. El parámetro *fixedRate* indica el período que habrá entre cada ejecución del método. En este caso son 86400000 milisegundos, que son los equivalentes a 24 horas es decir, la aplicación ejecutará el método cada día y siempre a la misma hora.

El algoritmo del método es sencillo:

1. Obtiene de la base de datos la última fecha en la que se actualizó correctamente.

2. Se comprueba la diferencia de tiempo entre la última actualización correcta y la fecha actual.

3. Se realiza una petición al API enviando como parámetro el número de días de diferencia entre la última actualización y la fecha actual.

4. El API externa devuelve una lista de ítems los cuales se deben parsear para añadir a la base de datos.

5. En el caso de que el ítem sea una serie, se hace una nueva petición al API externa. Esta vez se piden las temporadas y episodios de la serie, los cuales se parsean para añadir en la base de datos.

6. Una vez se hayan recorrido todos los ítems, se actualiza la fecha de actualización a la actual.

7.5 Sprint 4: Gestión de comentarios

7.5.1 Análisis

Tras incorporar las operaciones referentes a la búsqueda y obtención de ítems, el siguiente pasó fue la elaboración de las historias de usuario relacionadas con la gestión de comentarios. En las tablas 7.8, 7.9, 7.10 y 7.11 se recogen las historias de usuario implementadas en este sprint y su descomposición en tareas.

US09 - Escribir comentario en un ítem
Como usuario autenticado quiero escribir un comentario en un ítem
Tareas
Implementación de las entidades correspondientes
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del *frontend* del formulario de escritura del comentario
Implementación del diseño del frontend

Cuadro 7.8: Sprint 4: Escribir comentario en un ítem

US11 - Ver comentarios de un ítem
Como usuario quiero ver los comentarios de un ítem
Tareas
Implementación de las entidades correspondientes
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout del cajón comentarios
Implementación del diseño del frontend

Cuadro 7.9: Sprint 4: Ver comentarios de un ítem

US13 - Editar comentario de un ítem
Como usuario autenticado quiero poder editar mis comentarios realizados en un ítem
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Implementación de la funcionalidad

Cuadro 7.10: Sprint 4: Editar comentario de un ítem

US14 - Borrar comentario de un ítem
Como usuario autenticado quiero poder borrar mis comentarios realizados en un ítem
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Implementación de la funcionalidad

Cuadro 7.11: Sprint 4: Borrar comentario de un ítem

7.5.2 Diseño e implementación

En esta iteración se creó la entidad *comment*. Al igual que en el anterior *sprint*, los servicios relacionados con la gestión de comentarios irán destinados al *CatalogService*, ya que los comentarios dependen completamente de los ítems del catálogo.

La creación de un comentario no es más que una petición POST al API REST de la aplicación en la cual el texto del comentario se envía en el cuerpo de la petición. Este texto ya se valida en el *frontend* de la aplicación, para que el usuario no pueda generar comentarios sin texto. Una vez llega la petición al controlador, este llama al servicio correspondiente el cual a través de la clase *CommentDao* se comunicará con la base de datos, primero para verificar si el usuario que realiza el comentario y el ítem sobre el cual se realiza exisren y después para guardar el comentario.

La edición de un comentario es similar a la creación de éste, sin embargo ahora se comprueba que el comentario a editar existe en la base de datos y su usuario e ítem se corresponden con los enviados en la petición. Cabe destacar que la edición de un comentario sólo será posible si el comentario no tiene ningún voto.

Tantoel campo donde se realiza el comentario como el de edición de un comentario (*TextField* componente de Material UI), tiene un contador con los caracteres restantes que el usuario puede poner. Para implementarlo se creó un estado en el componente (figura 7.10).

```
1  const [text, setText] = useState("");
```

Figura 7.10: Creación de un estado en el componente

Como se puede ver en la figura 7.11 a través de la *prop onChange* se modifica el estado del componente cada vez que el usuario vaya escribiendo. Después, con la *prop helperText* se muestra la longitud de la cadena de texto que se guarda en el estado del componente. Además con la *prop inputProps* se le puede pasar propiedades características de un input normal (en

este caso la longitud máxima).

```
<TextField
        inputProps={{
          maxLength: 500,
        }}
        className={classes.textField}
        fullWidth
        id="outlined-textarea"
        placeholder="New comment"
        value={text}
        disabled={editing}
        variant="outlined"
        onChange={(e) => setText(e.target.value)}
        helperText={`${text.length}/500`}
        onKeyUp={(event) => {
          if (event.key === "Enter") {
            dispatch(catalog.actions.makeComment({ catalogItemId: catalogItem.catalogItemId, text: text }));
            setMakeComment(true);
            setText("");
          }
        }}
      />
```

Figura 7.11: Propiedades del componente TextField

Para ver los comentarios se implementó el componente *CommentBox* (figura 7.12). Este componente tiene 3 partes:

- **NewCommentBox:** Implementa el *TextField* para la creacion de un comentario y los botones de Cancelar y Comentar

- **Zona de comentario:** En esta parte se renderizarán hasta un máximo de 10 comentarios. Para cada comentario muestra el nombre del usuario que lo realizó, el comentario, la fecha de creación (o modificación), los botones de edición y voto y la puntuación del comentario

- **Pagers:** Los pager son dos botones que implementan las llamadas necesarias al API para pedir la página siguiente o anterior de comentarios

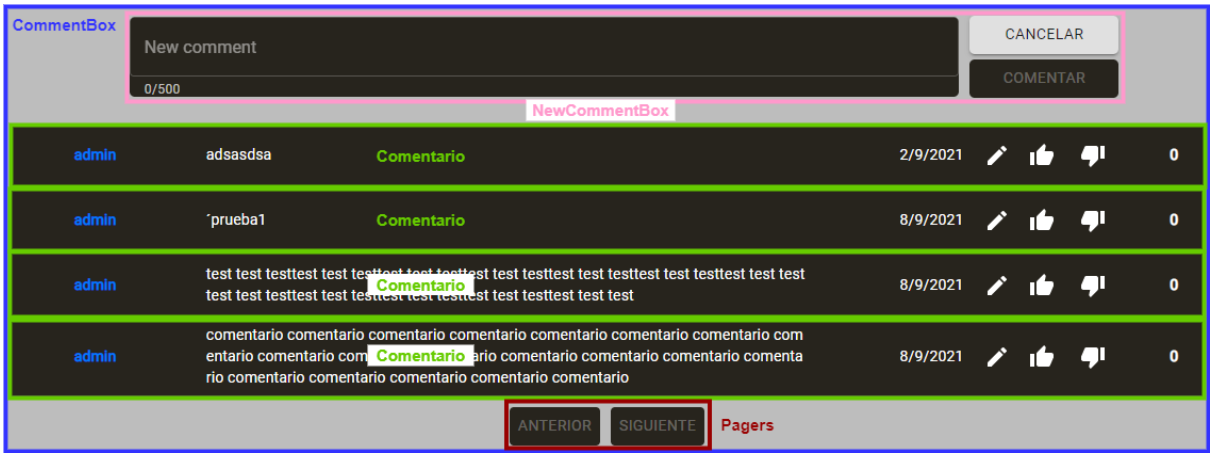

Figura 7.12: Componente *CommentBox* en la vista de detalles de un ítem

Si se hace clic en el botón del lápiz, se entrará en el modo edición del comentario (figura 7.13). Los botones de editar, voto positivo y voto negativo se cambian por el de eliminar comentario, cancelar edición y confirmar edición. Además, al editar el comentario, aparecerá un mensaje indicativo de que fue editado.

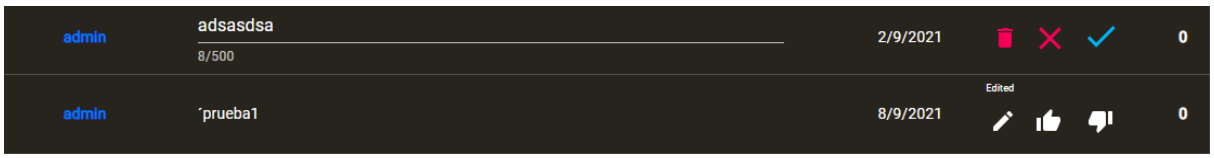

Figura 7.13: Comentario después de hacer clic en el botón de editar

7.6 Sprint 5: Gestión de puntuaciones

7.6.1 Análisis

Una vez que la aplicación ya permite las visualización de ítems y de comentarios, se procede a implementar los servicios relacionados con la gestión de puntuaciones; tanto de un ítem como de un comentario. En las tablas 7.12 y 7.13 se recogen las historias de usuario que se implementaron en el sprint y sus tareas.

US10 - Puntuar un ítem
Como usuario autenticado quiero puntuar un ítem
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout de la zona de votación
Implementación de la funcionalidad

Cuadro 7.12: Sprint 5: Puntuar un ítem

US12 - Puntuar comentario de un ítem
Como usuario autenticado quiero puntuar un comentario en un ítem
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout de la zona de votación
Implementación de la funcionalidad

Cuadro 7.13: Sprint 5: Puntuar comentario de un ítem

7.6.2 Diseño e implementación

Para la implementación de estas dos historias de usuario se tuvo que añadir dos entidades nuevas al modelo de datos. La primera, *Score* que representa la puntuación que un usuario da a un ítem. La segunda, *CommentScore*, representa la puntuación que un usuario le da a un comentario.

Aunque ambas entidades son puntuaciones, se decidió que las valoraciones de un ítem estarían en una escala de 1 a 10. Sin embargo para los comentarios el voto puede ser *like* o *dislike*. Esto lo hace asemejarse a otro tipo de plataformas en las que los comentarios se valoran de igual manera, por lo que para el usuario será más intuitivo.

Además, se modifican las entidades *CatalogItem* y *Comment*, a la que se le añadieron distintos atributos para facilitar las operaciones del cálculo de la media y la visualización de su puntuación.

- **CatalogItem:** A la entidad *CatalogItem* se le añaden dos atributos nuevos: *totalScore* y *numberOfVotes*. El primero guardará la suma total de todos los votos efectuados en el ítem, y el segundo, el número de votos. De esta forma, para calcular la puntuación

media de un producto no se tienen que obtener los votos efectuados sobre él, sino que, sencillamente cuando se añade un voto a la base de datos, también se modifica ese ítem.

En el caso de editar o eliminar un voto, como un usuario sólo puede tener un voto sobre el mismo ítem a la vez, se puede obtener fácilmente el valor del voto que emitió ese usuario sobre el ítem; y a la hora de modificar en la base de datos ese valor, también se modifica la *totalScore* y el *numberOfVotes* (en caso de eliminar un voto) del ítem.

- **Comment:** A la entidad *Comment* también se le añaden dos atributos nuevos: *positiveScore* y *negativeScore*, ya que los comentarios no muestran su puntuación media, sino la diferencia entre *positiveScore* y *negativeScore*. Al igual que con la entidad *CatalogItem*, a la hora de efectuar un voto sobre un comentario, aparte de añadir el voto a la base de datos, también se modifica el propio comentario.

  En el caso de editar o eliminar un voto, el funcionamiento es el mismo que con el *CatalogItem*: se obtiene el valor del voto en cuestión (ya que un usuario sólo puede emitir un voto por comentario a la vez) y se modifica tanto el voto como el comentario según proceda.

Como se puede ver en la figura 7.14, la zona de puntuaciones de un ítem muestra la puntuación media del ítem, la dada por el usuario y la zona de voto, esta última compuesta por el componente *Rating* de Material UI. Como la puntuación que se le puede dar a un ítem abarca de 1 a 10, al principio se optó por poner 10 estrellas, pero el feedback recibido por parte de los usuarios que probaron la aplicación indicaba que eran demasiadas, por lo que se optó por poner 5 estrellas permitiendo que se vote ”media” estrella para que se ajuste a la escala del voto.

Figura 7.14: Zona de puntuaciones de un ítem

En la figura 7.15 se puede comprobar cómo la zona para votar un comentario es diferente a la de un ítem. En vez de usar el componente *Rating* se usan dos botones. Estos botones cambiarán su color dependiendo de si el usuario votó positiva o negativamente.

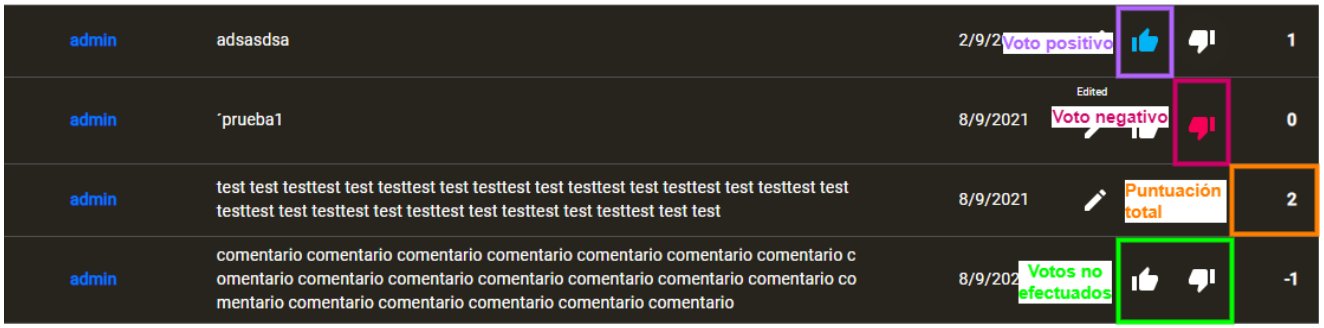

Figura 7.15: Zona de puntuación de un comentario

7.7 Sprint 6: Seguimiento de los ítems

7.7.1 Análisis

Para este sprint se decidió unicamente llevar a cabo una historia de usuario, la referente al seguimiento de los ítems. En la tabla 7.14 se recoge la historia de usuario y su descomposición en subtareas.

US15 - Marcar ítems para seguir
Como usuario autenticado quiero poder llevar un seguimiento de ítems que tengo para ver, las que estoy viendo y las vistas
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout de la zona de seguimiento
Implementación de la funcionalidad

Cuadro 7.14: Sprint 6: Marcar ítems para seguir

7.7.2 Diseño e implementación

Para implementar esta historia de usuario fue necesario añadir una entidad nueva al modelo: *User_CatalogItem*. Esta entidad actúa como tabla intermedia entre las entidades de *User* y *CatalogItem* y tiene un atributo *state* el cual para cada relación de usuario e ítem almacena el estado (*Para ver, Viendo* o *Visto*).

Como se ve en la figura 7.16, en la página de detalles de un ítem se mostrará un grupo de botones los cuales permitirán al usuario indicar si tiene pendiente de ver ese ítem, lo está

viendo o ya lo vio. En el caso de la figura está seleccionado el estado de ”Visto”.

Figura 7.16: Botones para la selección del estado de un ítem

7.8 Sprint 7: Interacción entre usuarios

En este *sprint* se implementan las últimas funcionalidades de la aplicación, relacionadas con la interacción entre usuarios y el perfil de estos. Estas se detallan en las tablas 7.15, 7.16, 7.17, 7.18, 7.19 y 7.20.

7.8.1 Análisis

US16 - Ver detalles de un usuario
Como usuario quiero ver los detalles de otros usuarios
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del perfil de usuario
Implementación de la funcionalidad

Cuadro 7.15: Sprint 7: Ver detalles de un usuario

US17 - Seguir a un usuario
Como usuario autenticado quiero poder seguir a otros usuarios
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Implementación de la funcionalidad

Cuadro 7.16: Sprint 7: Seguir a un usuario

US18 - Dejar de seguir a un usuario
Como usuario autenticado quiero poder dejar de seguir a otros usuarios
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Implementación de la funcionalidad

Cuadro 7.17: Sprint 7: Dejar de seguir a un usuario

US19 - Ver usuarios seguidos
Como usuario quiero poder ver a los usuarios seguidos
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout de la zona de usuarios seguidos
Implementación de la funcionalidad

Cuadro 7.18: Sprint 7: Ver usuarios seguidos

US20 - Ver comentarios realizados por un usuario
Como usuario quiero poder ver los comentarios de un usuario en concreto
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout de la zona de comentarios
Implementación de la funcionalidad

Cuadro 7.19: Sprint 7: Ver comentarios realizados por un usuario

US21 - Ver ítems seguidos por el usuario
Como usuario autenticado quiero poder ver los ítems que tengo en seguimiento
Tareas
Implementación del servicio correspondiente
Implementación de las peticiones en el controlador correspondiente
Diseño del layout de la zona de seguimiento
Implementación de la funcionalidad

Cuadro 7.20: Sprint 7: Ver ítems seguidos por el usuario

7.8.2 Diseño e implementación

Para implementar el seguimiento entre usuarios, se tuvo que añadir el atributo *followedUsers* a la entidad *User*. En la base de datos se corresponde con una tabla con dos columnas, una para el id del usuario y la otra para el id del usuario al que sigue.

En la figura 7.17 se muestran los datos que el propio usuario puede ver en su perfil. Un usuario podrá revisar el tiempo que ha pasado viendo ítems del catálogo, los usuarios a los que sigue y los 10 géneros que más visualiza, incluyendo el número de veces que ha visto

ítems pertenecientes a ese género.

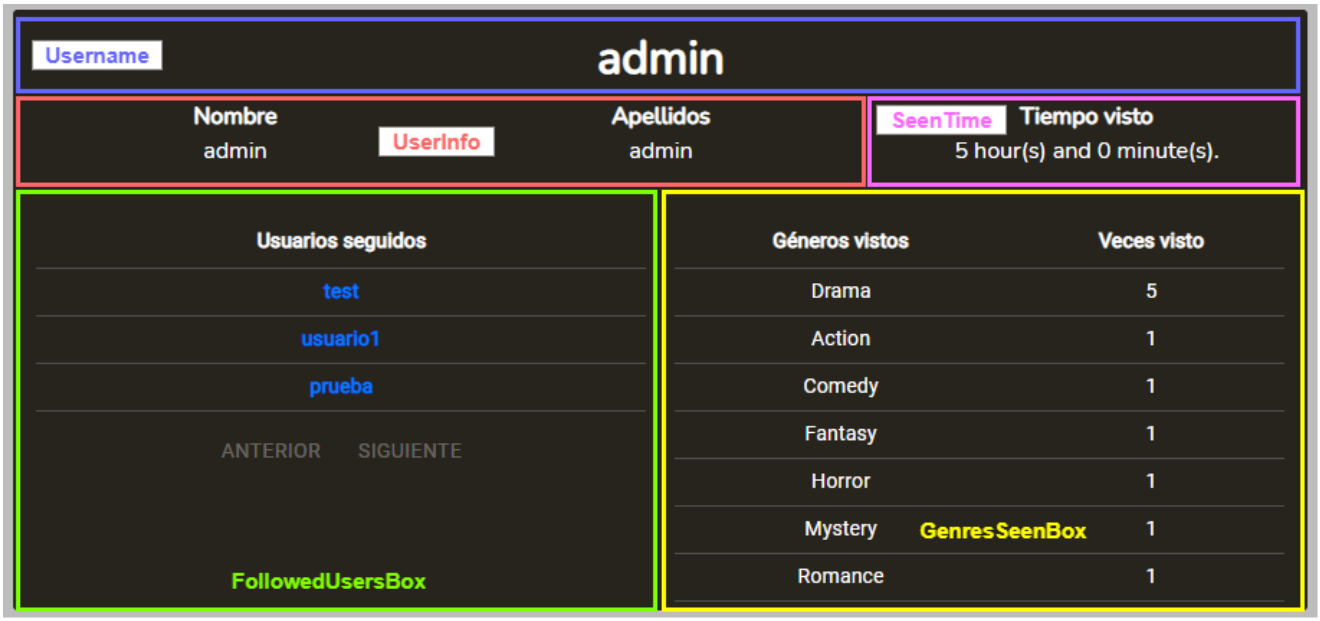

Figura 7.17: Información del usuario

Sin embargo, si el usuario se encuentra en el perfil de otra persona, para dar algo de privacidad, sólo se mostrará el nombre de usuario, el botón de seguir (o dejar de seguir), nombre y apellidos y el listado de usuarios a los que sigue (figura 7.18).

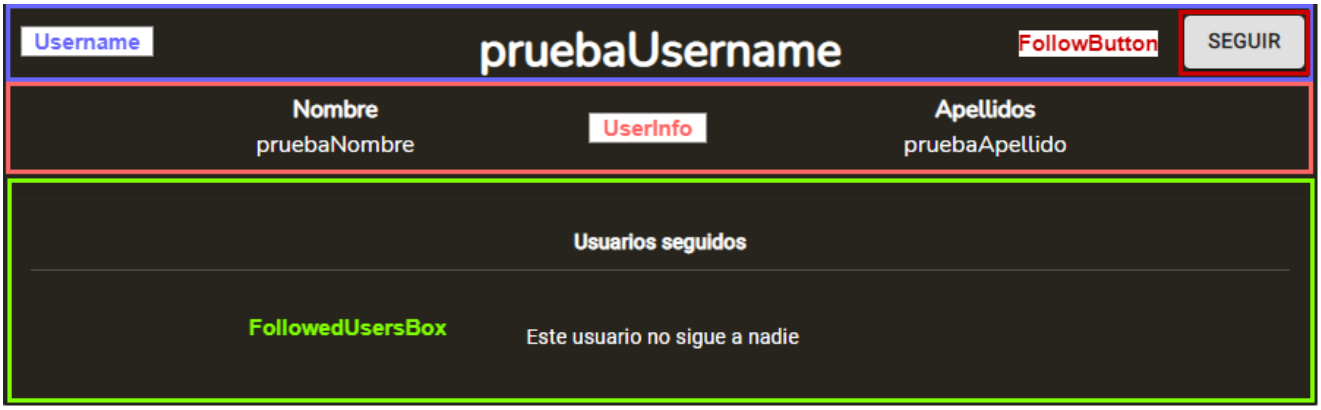

Figura 7.18: Información de otro usuario

En la parte de abajo de su propio perfil, el usuario podrá ver los ítems que tiene marcados ”Para ver”, ”Viendo” o ”Vistas”, los cuales se mostrarán en un carrusel y los comentarios realizados por el usuario, con un enlace al ítem en el cual se realizó dicho comentario (figura 7.19). Sin embargo si el perfil visualizado es el de otro usuario, por privacidad, no se mostrarán los

ítems seguidos.

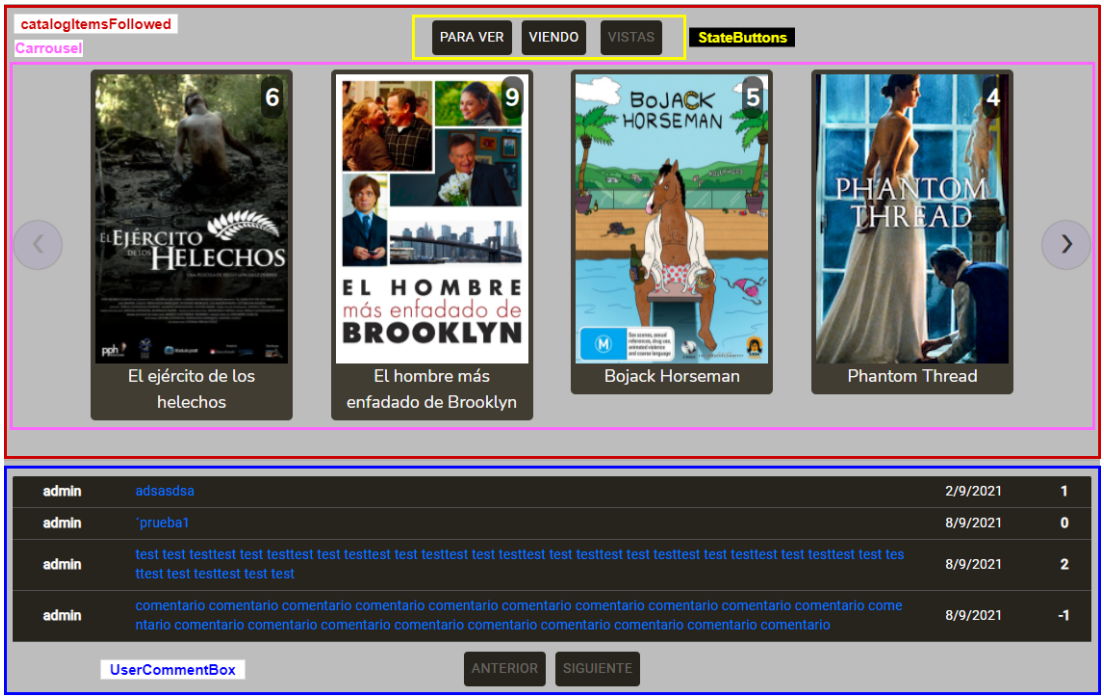

Figura 7.19: Información del usuario (ítems seguidos y caja de comentarios)

Al finalizar este *sprint*, se completaron todas las historias de usuario del *Product Backlog*.

7.9 Sprint 8: Arreglo de bugs y mejoras

En esta iteración diferentes usuarios externos al equipo de desarrollo realizaron distintas pruebas funcionales del sistema para comprobar si la aplicación cumplía el ser intuitiva y sencilla de usar para el usuario. Gracias al feedback recibido, se llevaron a cabo diferentes mejoras sobretodo de *user experience* y se corrigieron diferentes errores que se observaron con la aplicación terminada.

Capítulo 8

Conclusiones y trabajo futuro

8.1 Conclusiones

Una vez acabado el proyecto se considera que los objetivos planteados inicialmente han sido cumplidos.

Por un lado, se desarrolló una aplicación web completamente funcional que permite la puntuación y el seguimiento de las películas y series de los principales servicios de streaming, comentar y puntuar los comentarios que los usuarios realicen sobre los diferentes ítems del catálogo y la visualización de diferentes estadísticas por parte de los usuarios.

Otro objetivo del proyecto era el de construir una aplicación SPA rápida, intuitiva y agradable para el usuario. Para esto, diferentes usuarios llevaron a cabo pruebas funcionales del sistema. En algunas de las pruebas se dejaba al usuario navegar por la aplicación sin ningún objetivo concreto, mientras que en otras se le pedía que realizara alguna acción en particular para observar si podía lograr su objetivo sin ningún contratiempo.

La realización de este proyecto también sirvió para afianzar y potenciar los conocimientos obtenidos durante la carrera, sobre todo en la parte del *backend*, ya que durante las prácticas curriculares el alumno trabajó como *frontend developer* en proyectos que usaban los frameworks de React y Redux, por lo que en lo que al *frontend* de la aplicación se refiere, la única novedad fue el uso del framework de Material UI.

Las dos cosas que más se podrían destacar de lo que se aprendió durante la realización del proyecto, serían:

- **La profundización en el framework Spring Boot:** Aunque durante la carrera diversas asignaturas ya incorporan Spring Boot en sus prácticas, para el proyecto se tuvo que explotar aun más el potencial de este framework. Por ejemplo: la posibilidad de

ejecutar métodos periódicamente con la anotación *@Scheduled* o el uso de consulttas nativas desde un Dao.

- **La mejora en la capacidad encontrar errores y solucionarlos:** Algo a lo que se le da mucha importancia durante el grado es a depurar el código usando las herramientas de depuración, pero muchas veces por falta de habilidad en su uso o por el desconocimiento de su potencial, se termina depurando mediante la impresión de mensajes en la consola. Durante el proyecto fue crítico el uso de estas herramientas para poder solucionar los errores que iban surgiendo de manera óptima.

8.2 Trabajo futuro

Aunque el proyecto haya cumplido con los objetivos propuestos, hay varias funcionalidades que podrían ser interesantes implementar en la aplicación:

- **Mayor interacción entre usuarios:** La interacción de usuarios actualmente en la aplicación es básica, limitándose a seguir a otros usuarios y a puntuar comentarios. Una forma de mejorar esta interacción sería añadir mensajería privada a la aplicación, de forma que los usuarios puedan comunicarse entre ellos.
- **Listas de reproducción:** Actualmente los usuarios pueden marcar los ítems como ”Para ver”, ”Viendo” o ”Vistos” sin embargo, la creación de listas de reproducción podría ser una función interesante, por ejemplo creando listas de reproducción de las películas favoritas o sagas, dando la opción de compartir estas listas de reproducción con otros usuarios.
- **Más filtros de búsqueda:** Actualmente la aplicación sólo contempla buscar por el título del ítem, pero se podrían añadir más filtros de búsqueda como por ejemplo buscar por el género, por la duración, por la compañía de streaming…

Apéndices

Lista de acrónimos

API Application Programming Interface. 2

DOM Document Object Model. 33

IDE Integrated Development Environment. 36

JS JavaScript. 32

JVM Java Virtual Machine. 30

JWT Jason Web Token. 44

npm Node Package Manager. 35

POM Project Object Model. 30

REST Representational State Transfer. 2

SCM Source Code Management. 36

SPA Single Page Application. 1

VOD Video On Demand. 1

VSCode Visual Studio Code. 35

Glosario

API Interfaz que define un conjunto de operaciones con el objetivo de facilitar la comunicación entre aplicaciones.. 1

backend Capa oculta de una aplicación que se encarga de la gestión de datos y la lógica de negocio.. 1

framework Estructura real o conceptual destinada a servir de soporte o guía para la construcción de un proyecto.. 1

frontend Capa visible de una aplicación con la que el usuario interactúa directamente.. 1

Mockup Modelo, normalmente a escala, del diseño visual de un sistema. 17

Bibliografía

[1] "Página web de IMDb." [En línea]. Disponible en: https://www.imdb.com

[2] "Página web de My Anime List." [En línea]. Disponible en: https://www.myanimelist.net

[3] "Página web de Rotten Tomatoes." [En línea]. Disponible en: https://www.rottentomatoes.com

[4] "Página web de Metacritic." [En línea]. Disponible en: https://www.metacritic.com

[5] "Scrum Guides." [En línea]. Disponible en: https://scrumguides.org

[6] "Página web oficial de MySQL." [En línea]. Disponible en: https://www.mysql.com/

[7] "Página web oficial de Java." [En línea]. Disponible en: https://www.java.com/es/

[8] G. K. Christian Bauer and G. Gregory, *Java Persistence with Hibernate*, 2nd ed. Manning, 2016.

[9] "Página web oficial de Maven." [En línea]. Disponible en: https://maven.apache.org/

[10] "Página web oficial de Spring Framework." [En línea]. Disponible en: https://spring.io/projects/spring-framework

[11] "Página web oficial de Spring Boot." [En línea]. Disponible en: https://spring.io/projects/spring-boot

[12] "Página web oficial de IntelliJ IDEA." [En línea]. Disponible en: https://www.jetbrains.com/es-es/idea/

[13] "Página web oficial de JetBrains." [En línea]. Disponible en: https://www.jetbrains.com/

[14] "Marketplace de plugins de JetBrains." [En línea]. Disponible en: https://plugins.jetbrains.com/

[15] "Página web del estándar ECMAScript 11." [En línea]. Disponible en: https://262.ecma-international.org/11.0/

[16] "Página web oficial de React." [En línea]. Disponible en: https://es.reactjs.org/

[17] "Página web oficial de Angular." [En línea]. Disponible en: https://angular.io/

[18] "Página web oficial de Vue.js." [En línea]. Disponible en: https://vuejs.org/

[19] "Página web oficial de Svelte." [En línea]. Disponible en: https://svelte.dev/

[20] "Página web oficial de Preact." [En línea]. Disponible en: https://preactjs.com/

[21] "Página web oficial de Ember." [En línea]. Disponible en: https://emberjs.com/

[22] "Repositorio de GitHub oficial de React." [En línea]. Disponible en: https://github.com/facebook/react

[23] "Página web oficial de Redux." [En línea]. Disponible en: https://es.redux.js.org/

[24] "Página web oficial de Flux." [En línea]. Disponible en: https://facebook.github.io/flux/

[25] "Página web oficial del *binding* de React y Redux." [En línea]. Disponible en: https://react-redux.js.org/

[26] "Página web oficial de npm." [En línea]. Disponible en: https://www.npmjs.com/

[27] "Página web oficial de Node.js." [En línea]. Disponible en: https://nodejs.org/es/

[28] "Página web oficial de MaterialUI." [En línea]. Disponible en: https://material-ui.com/

[29] "Página web oficial de Visual Studio Code." [En línea]. Disponible en: https://code.visualstudio.com/

[30] "Marketplace de Visual Studio Code." [En línea]. Disponible en: https://marketplace.visualstudio.com/vscode

[31] "Página web oficial de Git." [En línea]. Disponible en: https://git-scm.com/

[32] "Página web oficial de Trello." [En línea]. Disponible en: https://trello.com/es

[33] "Página web oficial de Postman." [En línea]. Disponible en: https://www.postman.com/

[34] "Página web oficial de DataGrip." [En línea]. Disponible en: https://www.jetbrains.com/es-es/datagrip/

[35] "Página del componente react-elastic-carousel." [En línea]. Disponible en: https://www.npmjs.com/package/react-elastic-carousel