UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Agenda para el seguimiento de contenidos adiovisuales.

Trabajo Fin de Grado

Grado en Ingeniería de Sistemas de Telecomunicación, Sonido e Imagen

AUTOR/A: Gómez Morales, Pedro

Tutor/a: Bataller Mascarell, Jordi

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITÈCNICA SUPERIOR DE GANDIA

Grado en Ing. Sist. de Telecom., Sonido e Imagen

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCOLA POLITÈCNICA SUPERIOR DE GANDIA

“Agenda para el seguimiento de contenidos audiovisuales”

TRABAJO FINAL DE GRADO

Autor/a:
Gómez Morales, Pedro

Tutor/a:
Bataller Mascarell, Jordi

GANDIA, 2022

Resumen

Las plataformas de streaming ponen a nuestra disposición grandes cantidades de contenido y, con los años, hemos visto cómo han ido evolucionando y surgiendo cada vez más plataformas de este tipo.

Esto ha dado lugar a una competitividad, ya que todas tratan de atraer suscriptores y la principal estrategia se basa en ofrecer un catálogo de contenido lo más amplio y exclusivo posible.

Al otro lado, los consumidores vemos exponencialmente incrementadas nuestras opciones de consumo. Esto puede ocasionar dificultades a la hora de llevar un seguimiento del contenido que hemos consumido o queremos consumir.

Por esta razón, nos proponemos a desarrollar una aplicación web que permita a los usuarios hacer un seguimiento del contenido que han visto y/o quieren ver. De esta manera no perderán tiempo buscando opciones ni recordando si han visto o no una película o serie.

Esta aplicación está desarrollada utilizando algunas tecnologías punteras del mercado actual, como es VueJS, un framework del lenguaje de programación JavaScript. Haremos consultas a una API para obtener datos sobre series y películas y utilizaremos una base de datos de MySQL para almacenar los datos necesarios. Gestionaremos dichos datos a través de Flask, un micromarco del lenguaje de programación Python.

Palabras clave: contenidos audiovisuales, aplicación cliente-servidor, bases de datos, página web, modelo-vista-controlador

Abstract

Streaming platforms have provided us with a huge amount of content. Over the years, we have seen how more and more platforms of this type have evolved and emerged.

This has led to fierce market competition, as they all try to attract subscribers, so the main strategy is based on offering the largest possible catalogue of exclusive content.

On the other hand, those subscribers see our consumption options increase exponentially. This can make it difficult to keep track of the content we have consumed or want to consume.

Due to this, we have set out to develop a web application that allows users to keep track of the content they have watched or would like to watch so they will not waste time searching for options or remembering whether they have watched already seen content.

We have developed this application using some of the latest technologies on the market today, such as VueJS, a framework for the JavaScript programming language. We will make queries to an API to obtain data about series and movies and we will use a MySQL database to store the necessary data. We will manage this data through Flask, a microframework of the Python programming language.

Keywords: audiovisual content, client-server application, databases, web page, model-view-controller

Contenido

Ilustraciones

1.- Introducción

1.1.- Presentación y objetivos

Las plataformas de streaming han cambiado la concepción que teníamos las personas sobre el consumo de entretenimiento audiovisual, poniendo a nuestra disposición grandes bibliotecas de contenido bajo demanda y haciendo posible que podamos consumirlo en cualquier momento y prácticamente desde cualquier sitio.

El éxito de estas plataformas ha sido de tal magnitud que, durante la última década, no solo han evolucionado las primeras que se popularizaron (Netflix, HBO...), sino que hemos visto surgir una gran variedad de ellas (Amazon Prime Video, Rakuten TV, Filmin, Disney+...).

Este hecho ha venido acompañado de una gran competencia de mercado entre las mismas, ya que cada plataforma trata de atraer al mayor número de suscriptores y la principal estrategia se basa en lanzar el mayor catálogo de contenido exclusivo posible.

Al otro lado, los consumidores de dicho contenido vemos incrementadas nuestras opciones de consumo de forma exponencial y, en ocasiones, puede resultar difícil para una persona recordar si ha visto o no una película o una serie.

Por esta razón, hemos decidido desarrollar una aplicación web que permite a los usuarios realizar un seguimiento del contenido audiovisual (series y películas) que hayan consumido o estén interesados en consumir.

Con esta aplicación, se pretende que los usuarios puedan:

- buscar series y películas en una base de datos online.
- anotar las series y películas, resultado de dichas búsquedas, en una suerte de agenda.
- establecer el estado de las series y películas anotadas como “vistas” o “pendientes de ver”.
- revisar el contenido completo añadido a sus perfiles.
- recibir recomendaciones no personalizadas de contenido.

Por último, cabe destacar que mis objetivos formativos consisten en aprender a utilizar herramientas que no he podido ver durante la titulación. Una de ellas es VueJS, un framework de JavaScript muy utilizado actualmente. En cuanto a lenguajes de programación, me interesa aprender Python porque facilita trabajar con muchos campos que están en auge en la actualidad. También estoy interesado en el manejo de APIs y bases de datos y, de paso, reforzar mis conocimientos sobre Git.

Por todas estas razones y algunas más que iré comentando a lo largo del documento, he decidido utilizar dichas tecnologías para el desarrollo de la aplicación.

1.2.- Estructura de la memoria

Esta memoria está organizada de la siguiente forma:

- En el capítulo 2, **Contexto de la aplicación**, explicamos todo sobre el ámbito en el que se va a desarrollar la aplicación, así como las tecnologías empleadas.

- En el capítulo 3, **Análisis de requerimientos**, definimos el alcance de la aplicación y las pantallas de esta.
- En el capítulo 4, **Diseño de la aplicación**, explicamos el diseño de nuestra aplicación de lo general a lo particular.
- En el capítulo 5, **Implementación**, describimos las metodologías usadas, así como la estructura de las carpetas, su contenido y los problemas que hemos enfrentado.
- En el capítulo 6, **Manuales**, detallamos cómo ejecutar la aplicación para su uso y mantenimiento.
- En el capítulo 7, **Evaluación**, evaluamos la aplicación desde el punto de vista de usabilidad y la comparamos con otras aplicaciones similares.
- En el último capítulo, presentamos las **conclusiones** y proponemos **ampliaciones futuras**, así como otras mejoras, tanto de estilo como de funcionamiento, que puedan hacer que nuestra aplicación compita en el mercado.

2.- Contexto de la aplicación

Durante la última década, se han popularizado una gran variedad de plataformas digitales de contenido **"*on demand*"** (bajo demanda), las cuales permiten que los usuarios suscritos puedan acceder a largas bibliotecas de contenido desde cualquier dispositivo inteligente (televisores, teléfonos, ordenadores, etc.) que, por lo general, esté conectado a Internet (algunas ofrecen la posibilidad de descargar dicho contenido para acceder a él de forma local, sin necesidad de estar conectado a la red). Estos servicios surgieron como una alternativa a la descarga ilegal de archivos, permitiendo a los usuarios disponer del contenido que prefieran pagando un precio justo por él y, además, ofreciendo como principal ventaja la posibilidad de acceder a dicho contenido sin ocupar espacio y desde cualquier parte.

Estas plataformas pertenecen a los llamados **servicios OTT** (*over-the-top*), que consisten en la transmisión de audio, vídeo y otros contenidos a través de Internet sin la implicación de los operadores tradicionales en el control o la distribución del contenido[1].

El contenido ofertado por estas plataformas puede ser muy variado, yendo desde cine hasta música y pasando por series de televisión, videojuegos e incluso libros. Sin embargo, las que más se han popularizado han sido las que ofrecen películas y series.

El éxito de estas plataformas ha sido de tal magnitud que, durante la última década, no solo han evolucionado las primeras que se popularizaron (como Netflix, que cuenta actualmente con más de 200 millones de suscriptores[25]), sino que hemos visto surgir una gran variedad de plataformas internacionales (HBO Max, Rakuten TV, Filmin, Disney+, Hulu, Paramount+, Apple TV+...) e incluso nacionales (Atresplayer del grupo Atresmedia, Mitele del grupo Mediaset, Movistar Plus de Movistar…).

Este hecho ha venido acompañado de una gran competencia de mercado entre las mismas, ya que cada plataforma trata de atraer al mayor número de suscriptores y la principal estrategia se basa en ofrecer el catálogo más amplio y exclusivo posible.

No solo compiten por hacerse con los derechos del contenido de otras productoras y proveedores, como es el caso de Disney+, que cuenta en exclusiva con franquicias como las de Disney, Pixar, Marvel, Star Wars y National Geographic. Sino que, además, también compiten por lanzar el mejor contenido autoproducido, como el caso de Netflix, que cuenta con las Netflix Originals, o Amazon Prime Video y sus Amazon Studios.

Prácticamente todos los meses se estrenan varias películas y series originales en cada plataforma. En el caso de Netflix, estrenó un total de 57 series, especiales, miniseries, documentales y películas en el año 2021[2]. Si pensamos en la cantidad de contenido que se ha podido estrenar en total entre todas las plataformas, los números son altísimos.

Al otro lado, los consumidores de dicho contenido vemos incrementadas nuestras opciones de consumo de forma exponencial y, en ocasiones, puede resultar difícil para una persona recordar si ha visto o no una película o una serie.

Es frecuente el caso en el que una persona comienza a ver una película en alguna de sus plataformas y, a mitad de esta, se da cuenta de que ya la había visto. O querer ponerse a ver una serie o una película en el único momento del día en que se dispone de tiempo, pero perderse en las inmensas bibliotecas de contenido y perder demasiado tiempo decidiendo qué ver.

Por esta razón, vamos a desarrollar una aplicación que permita a los usuarios hacer un seguimiento del contenido que han visto, así como del que quieren ver. De esta forma, no perderán tiempo buscando opciones ni recordando si han visto o no una película o serie.

2.1.- Aplicaciones similares

Existen ya varias aplicaciones de este estilo, cada una con sus particularidades, sus pros y sus contras. Una de las que podrían considerarse pioneras es **Tviso**, una plataforma que nació de series.ly, una antigua web ilegal que albergaba multitud de películas y series a las que se podía acceder gratuitamente a través de enlaces externos a webs de reproducción en streaming.

Tras la aprobación de la Ley Sinde (que prohíbe la circulación de material con derechos de autor como son las películas y las series), series.ly se vio obligada a retirar todos los enlaces que conducían a este tipo de contenido.

Tras esto, Tviso compró series.ly y se convirtió en un portal donde se recoge información sobre las series o películas que se estrenan o están disponibles en otras plataformas legales como HBO o Netflix[3].

Durante muchos años, Tviso ha sido una de las más utilizadas, ya que los usuarios que utilizaban series.ly siguieron en ella al conservarse guardado el contenido que han visto y/o quieren ver.

Sin embargo, en 2017 lanzaron Tviso TV, un TV Box pensado para competir contra el Apple TV, el Chromecast de Google o el Fire Stick de Amazon, pero la idea no termino de cuajar y no tuvo la necesaria salida comercial[4].

Este hecho hizo quebrar a Tviso, que anunció su cierre en 2019, dejando la plataforma de seguimiento de series y películas a la deriva, lo cual resulta poco confiable por varias razones:

- En ocasiones, la web cae durante días o incluso semanas.
- No admite nuevos registros, ya que su sistema de registro requiere de autenticación de correo electrónico, pero ese es un servicio que no está activo.
- Cuando funciona, depende demasiado de la comunidad, que es muy escasa.

Probablemente, su mayor ventaja es su simpleza, ya que se trata de una plataforma muy sencilla e intuitiva de manejar, ofreciendo muchas facilidades para buscar contenido y organizarlo.

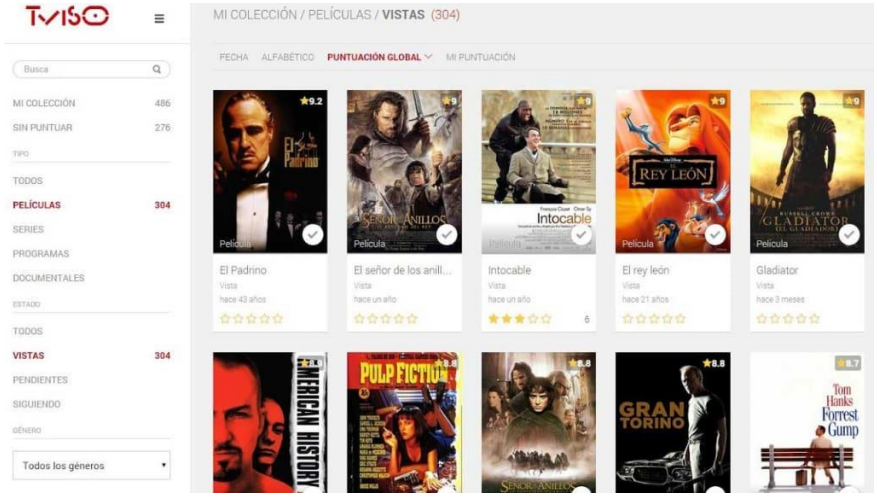

Ilustración 1: Tviso

Otra de las plataformas de seguimiento más utilizadas en la actualidad es **TV Time**, que surgió como una aplicación para teléfonos móviles y centros multimedia para que los usuarios pudieran llevar un seguimiento de las series que seguían. Más adelante, también llegó como versión web.

Esta aplicación ofrece grandes ventajas, como avisarnos cuando está disponible el próximo episodio de nuestras series. Además, también permite a los usuarios comentar cada episodio o serie. También permitió durante un tiempo que los usuarios pudieran grabar su reacción en vídeo mientras ven un episodio y lo pudieran compartir en la propia aplicación, pero esta función desapareció al poco tiempo de aparecer.

Al igual que Tviso, también destaca por su simpleza, además de por tener tanto una versión de escritorio como una aplicación para teléfonos móviles, ofreciendo la posibilidad de tenerlo todo mucho más a mano.

Su gran desventaja es que, por ahora, solo permite realizar un seguimiento de series, por lo que no cuenta con la posibilidad de buscar también películas.

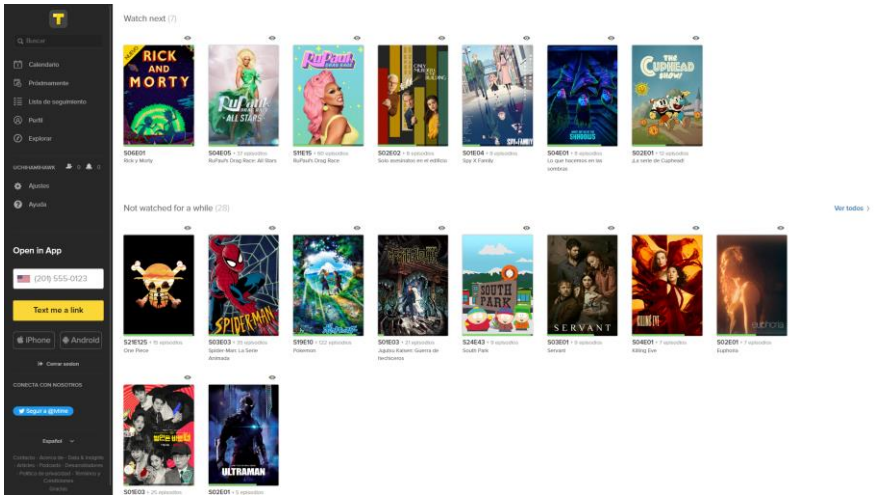

Ilustración 2: TV Time

Por último, vamos a hablar del portal **Palomitacas**, una plataforma desarrollada en España con un diseño bastante atractivo. Este portal funciona a modo de agenda con todo el contenido que vemos cada día. Podemos escribir críticas y permite trabajar como una improvisada red social.

Quizá este sea su punto más fuerte, ya que cuando tenemos amigos agregados, podemos ver qué está viendo cada uno, comparar el contenido visionado y otras cosas similares.

La plataforma también ofrece un ranking con el contenido que más se está viendo en todo momento, gracias a la participación de los propios usuarios. De esta manera, puede ser sencillo enterarse de nuevas series o películas.

No obstante, pensamos que este portal resulta muy poco intuitivo, sobre todo en lo respectivo a las series. Algunas funciones se encuentran algo escondidas y la navegación por el sitio web no resulta del todo agradable. Esto obliga a los usuarios a necesitar un tiempo de adaptación para aprender a usarla y puede ser que algunos no lleguen a quedarse en la plataforma porque les parezca tedioso al principio.

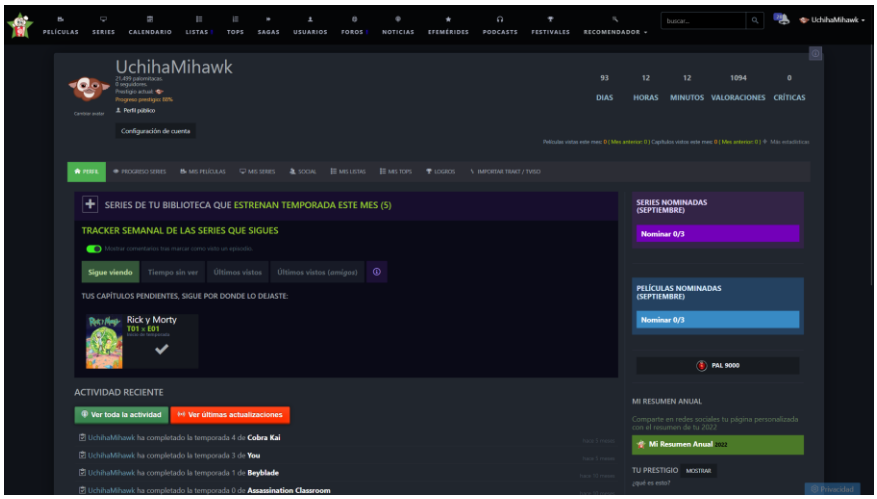

Ilustración 3: Palomitacas

3.- Análisis de requerimientos

Antes de comenzar con el diseño de la aplicación, se han establecido una serie de requisitos que reflejen el alcance del proyecto. Estos requisitos son los siguientes:

- Registro de usuarios.
- Inicio de sesión de usuarios.
- Mostrar películas y series mediante diferentes categorías como:
  - Populares.
  - En cines.
  - Mejor valoradas.
- Buscar películas, series y otros usuarios.
- Añadir películas y series a la biblioteca de contenido del usuario, permitiendo asignarles un estado:
  - Visto.
  - Pendiente de ver.
- Modificar el estado de las películas y series ya añadidas.
- Eliminar las películas y series de la colección.
- Visualizar la biblioteca de contenido del usuario.
- Ordenar la biblioteca de contenido del usuario según varios criterios:
  - Por título (ascendente y descendente).
  - Por tipo de contenido (ascendente y descendente).
- Filtrar la biblioteca de contenido del usuario según el estado:
  - Mostrar solo elementos vistos.
  - Mostrar solo elementos pendientes de ver.
- Acceder a una ficha con información ampliada del elemento (sinopsis, año de estreno, equipo, reparto, tráiler, etc.)
- Puntuar las series y películas, ofreciendo una valoración.
- Interactuar con otros usuarios.

La aplicación contará con las siguientes pantallas:

- **Pantalla 1:** Login para identificar o registrar al usuario
  - Dos cuadros de texto (usuario, contraseña)
  - Dos botones (iniciar, registrarse)
- **Pantalla 2:** Pantalla principal
  - Películas pendientes del usuario
  - Series pendientes del usuario
  - Películas populares
  - Series populares
  - Películas en cines
- **Pantalla 3:** Pantalla de búsqueda
  - Cuadro de texto para introducir el nombre de la película o serie a buscar
  - Los resultados de la búsqueda
- **Pantalla 4:** Perfil de usuario
  - Series y películas vistas y/o pendientes de ver.
  - Filtros para ordenar el contenido por título y tipo de contenido.
- **Pantalla 5:** Fichas
  - Portada
  - Título y año de lanzamiento.

    - Sinopsis.
    - Botones para añadir como visto o pendiente de ver.
    - Tráiler
    - Cuadro de texto para comentarios y valoraciones.

El flujo de navegación entre las diferentes pantallas es totalmente flexible una vez iniciada la sesión del usuario:

- La primera pantalla a la que se accede será la para registrarse y/o iniciar sesión. Al iniciar sesión, la aplicación redirigirá al usuario a la pantalla principal de forma automática.
- Desde la pantalla principal (y cualquiera de las otras a excepción de la pantalla de autenticación), el usuario podrá:
    - Ir a la pantalla de búsqueda pulsando el botón de búsqueda.
    - Ir a la pantalla de perfil pulsando el botón de perfil.
    - Cerrar sesión pulsando el botón de logout. Esta acción redirigirá al usuario a la pantalla de autenticación.

4.- Diseño de la aplicación

4.1.- Diseño de las pantallas

Para que el usuario pueda interactuar con la base de datos, serán necesarias una serie de pantallas que cuenten con una serie de componentes para facilitar la comunicación con el servidor:

- En una pantalla, que llamaremos **pantalla de autenticación**, se debe poder registrar usuarios e iniciar sesión. Esta pantalla tendría que contener dos cuadros de entrada de texto, uno para el nombre de usuario y otro para la contraseña. Además, un botón para cada una de las funciones mencionadas: registro de usuarios e inicio de sesión.

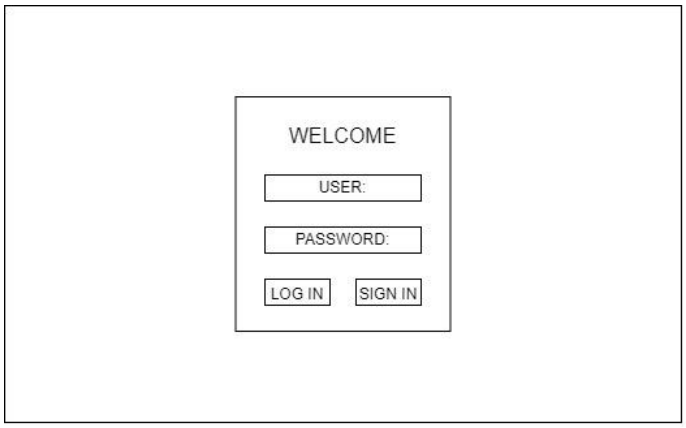

Ilustración 4: Diseño de la pantalla de autenticación

- Tendremos una **pantalla principal**, en la que mostraremos una recopilación con el contenido que el usuario ha marcado como pendiente de ver, ya que consideramos que es la opción que más a mano debería tener el usuario. Sería interesante añadir en esta parte más elementos, como recomendaciones no personalizadas, películas populares, series del momento, las mejor valoradas, etc.

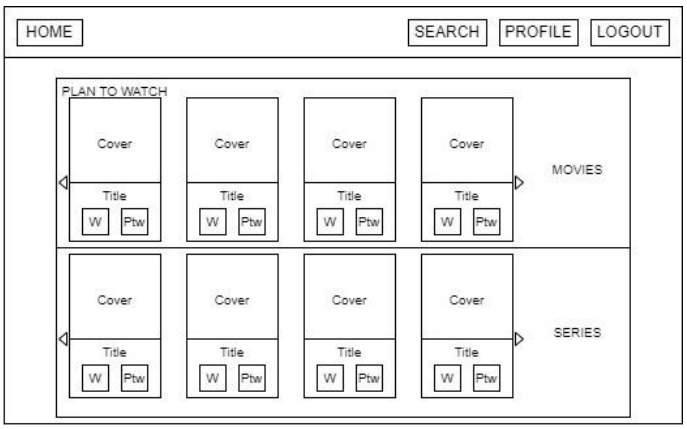

Ilustración 5: Diseño de la pantalla principal

- Para realizar **búsquedas** se necesitará de otra pantalla. Los requisitos mínimos de la misma serán un cuadro de entrada de texto para que el usuario introduzca el nombre de la película o serie que quiere buscar y que se muestren los resultados de dicha búsqueda en pantalla. De esta manera, el usuario puede añadir contenido a su perfil.

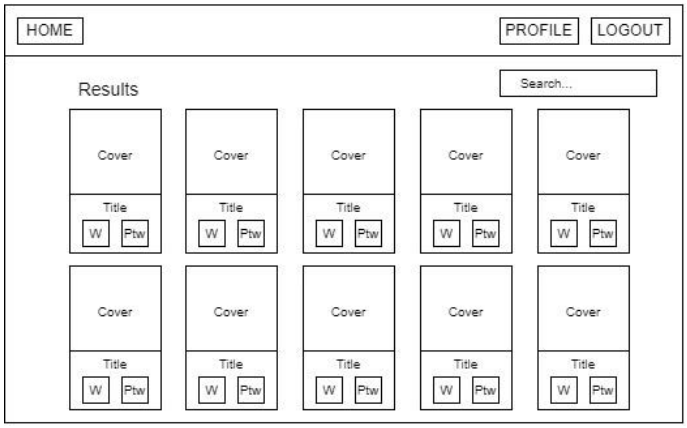

Ilustración 6: Diseño de la pantalla de búsqueda

- Para visualizar todo el contenido almacenado por el usuario, necesitaremos una **pantalla de perfil**, en la que se muestren todas las series y películas, vistas y pendientes de ver, que el usuario ha ido anotando en su perfil. También podrá tener la opción de ordenar el contenido alfabéticamente o filtrarlo según el tipo y el estado.

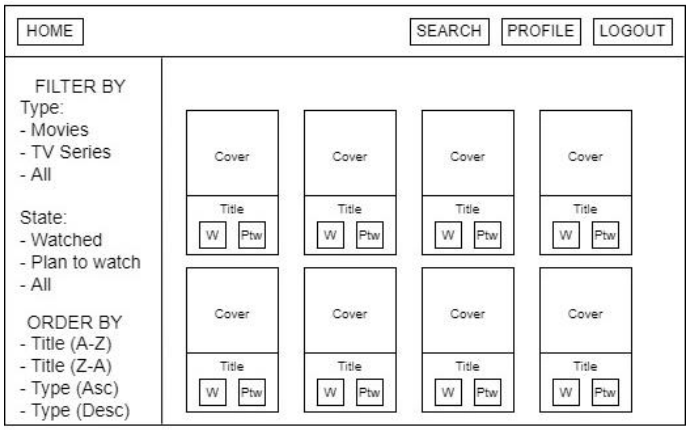

Ilustración 7: Diseño de la pantalla de perfil

Con estas cuatro pantallas, habríamos cubierto la funcionalidad más básica de la aplicación: hacer seguimiento del contenido que hemos visto o queremos ver.

4.2.- Arquitectura de la aplicación

En software, se suele seguir un **patrón arquitectónico** para estructurar el diseño y el desarrollo de la aplicación. Estos patrones nos permiten tener una guía de la estructura general de nuestro software, de manera que nos sea más fácil probarlo, escalarlo y mantenerlo.

Existen varios patrones arquitectónicos a la hora de desarrollar software. En este caso, vamos a utilizar el **patrón MVC** (Modelo-Vista-Controlador), ya que nos permite separar la lógica de negocio de las interacciones con el usuario[5].

Este patrón se divide en tres capas:

- **Vista:** como su propio nombre indica, es la capa relacionada con las pantallas, lo que el usuario ver mientras navega por la aplicación.

- **Modelo:** es la capa de datos, encargada de administrar la lógica de negocio y la conexión con la base de datos u otros servicios.
- **Controlador:** es la capa encargada de actualizar la información del modelo y, en consecuencia, de la vista.

El usuario va a provocar una serie de eventos conforme navega por la aplicación. Estos eventos van a ser recibidos por el controlador, que será quien los gestione. Para ello, enviará una petición al modelo, que hará las acciones necesarias para responder a dicha petición. Una vez el modelo termine su función, se actualizará la vista según los resultados obtenidos y se le mostrará al usuario.

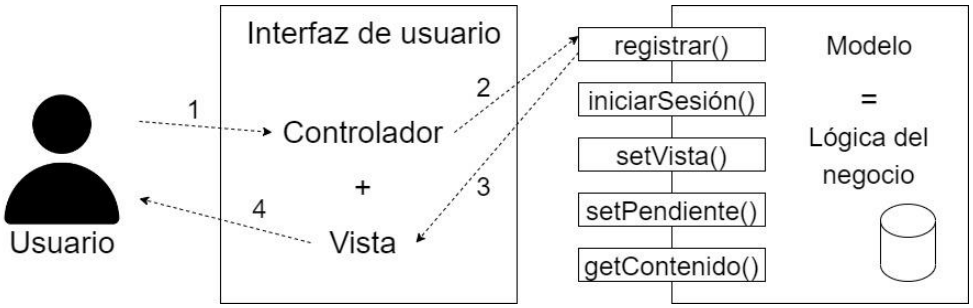

Ilustración 8: Modelo-Vista-Controlador

Para nuestra aplicación, dispondremos en el lado del cliente de una serie de funciones que tendrán como única función comunicarse con la lógica del negocio del lado del servidor utilizando una biblioteca, y será este quien se encargará de realizar las acciones necesarias.

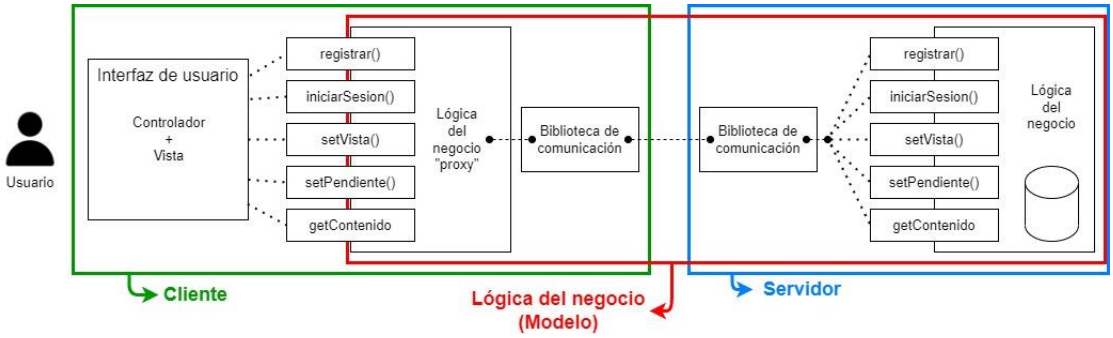

Ilustración 9: MVC aplicado a nuestra aplicación

El usuario, a través del controlador, va a provocar que se ejecuten los métodos de la lógica del negocio "proxy" conforme navegue. Estos métodos van a encargarse únicamente de realizar peticiones a la lógica del negocio del lado del servidor, donde tendremos métodos con el mismo nombre que se realizarán las acciones pertinentes. Una vez realizadas dichas acciones, el modelo se comunicará de vuelta con el cliente, haciendo que el controlador actualice la vista.

Por ejemplo, si un usuario trata de iniciar sesión en su perfil, introducirá sus datos (usuario y contraseña) y pulsará el botón correspondiente para iniciar sesión. Este botón ejecutará un método en la lógica del negocio "proxy" que simplemente utilizará una biblioteca de comunicación para realizar una petición al servidor. Dicha petición provocará que se ejecute el método correspondiente al inicio de sesión de la lógica del negocio del lado del servidor, que accederá a la base de datos para comprobar que los datos introducidos por el usuario son correctos. Una vez comprobado, el servidor enviará al cliente una respuesta que el controlador utilizará para actualizar la vista que se le muestra al usuario.

4.3.- Diseño del backend

En esta parte tendremos todo lo correspondiente a la lógica del negocio por parte del servidor. Será la parte encargada de recibir peticiones del cliente y ejecutar alguno de los métodos en función de la petición recibida.

Los principales métodos que deberemos tener son los siguientes:

- **Método para dar de alta un usuario:** este método deberá recibir el nombre de usuario y la contraseña establecidos por el usuario. Deberá comprobar en la base de datos que el nombre de usuario no exista y, si se cumple esta condición, añadirlo. Si el usuario ya existe, deberá enviar un mensaje de error al frontend.
- **Método para iniciar la sesión de un usuario:** similar al método anterior, deberá recibir el nombre de usuario y la contraseña y comprobar en la base de datos que dichos datos recibidos sean correctos.

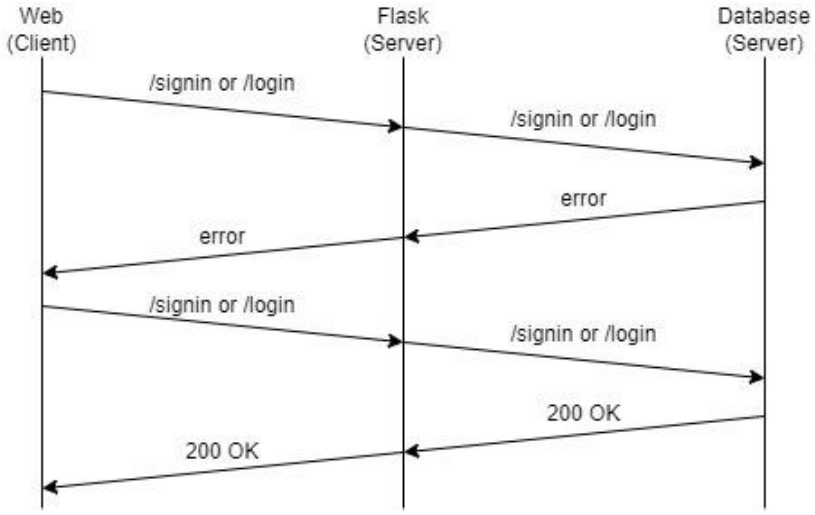

Ilustración 10: Mensajes cliente-servidor al registro e inicio de sesión de usuario

- **Método para anotar una película o serie como vista:** deberá recibir datos sobre la película o serie en sí (el identificador, el tipo y el estado), así como el identificador del usuario que quiere añadirla. Aquí habrá tres opciones: si la película o serie ya había sido añadida como vista, se eliminará de la base de datos; si había sido añadida como pendiente de ver, se cambiará su estado a vista; y si no estaba añadida, se añadirá como vista.
- **Método para anotar una película o serie como pendiente de ver:** el funcionamiento de este método sería exactamente igual que el método anterior, cambiando el estado de vista por el de pendiente de ver.

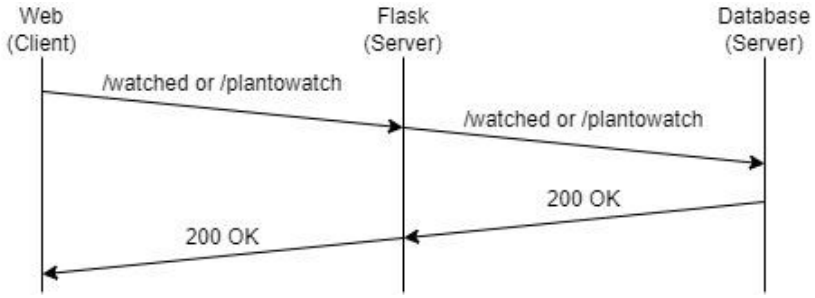

Ilustración 11: Mensajes cliente-servidor al anotar una serie o película

- **Método para obtener todo el contenido almacenado en la base de datos:** este método se utilizará para recuperar de la base de datos todo el contenido que ha almacenado el usuario con la sesión activa. Recibirá en la petición el identificador del usuario y enviará al cliente una lista con todo el contenido de dicho usuario.

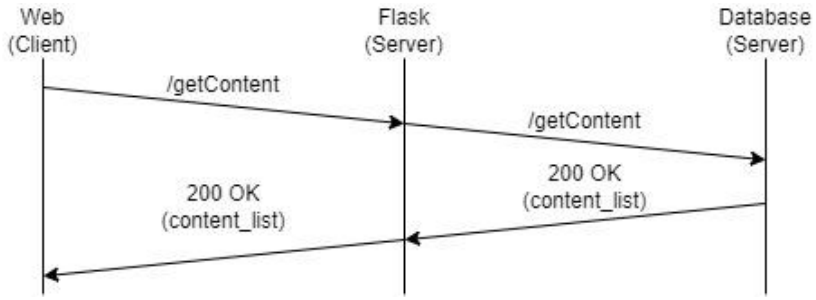

Ilustración 12: Mensajes cliente-servidor al recuperar todo el contenido del usuario

4.4.- Base de datos

La base de datos será relacional y estará formada, en principio, por dos tablas: una tabla para llevar un registro de los usuarios y otra tabla para llevar un registro del contenido registrado por todos ellos. A medida que la aplicación crezca, se podrían ir añadiendo más tablas que nos permitan realizar más funciones.

Llamaremos a la primera tabla con el nombre de **user**. En esta tabla, almacenaremos datos referentes al registro de los usuarios, como sus nombres de usuario, sus contraseñas (que serán datos de texto), y un identificador numérico único asociado a cada usuario.

Por otro lado, habrá una tabla para el contenido, a la que llamaremos **content**, en la cual se registrará el contenido almacenado en los perfiles de usuario, es decir, las películas y series que añadan a su biblioteca. Esta tabla dispondrá, en principio, de cuatro columnas: una para almacenar el identificador del elemento proveniente de la API de IMDB, otro con el tipo de elemento (serie o película), otro con el estado marcado por el usuario (visto o pendiente de ver) y otro con el identificador del usuario que lo ha añadido. También sería interesante almacenar algunos atributos básicos de los elementos recogidos de la API, como títulos o imagen de portada. De esta manera, evitaremos tener que hacer demasiadas peticiones a la API en determinadas ocasiones.

5.- Implementación

En este capítulo vamos a comentar todo lo relacionado con el proceso de desarrollo de la aplicación, empezando por comentar la metodología de trabajo seguida y las herramientas de desarrollo, siguiendo con la estructura que se ha seguido para organizar el código y terminaremos comentando alguno de los problemas resueltos durante la implementación.

5.1.- Metodología de trabajo y herramientas de desarrollo

Durante el desarrollo de la aplicación, hemos utilizado la metodología de **desarrollo en cascada**, un procedimiento secuencial que se caracteriza por dividir los procesos de desarrollo en sucesivas fases de proyecto[6].

En este caso, hemos dividido el proceso en las cinco fases siguientes:

1. **Análisis**. Hemos planificado superficialmente el proyecto, anotando una lista de requisitos y analizando qué tecnologías podrían ser de utilidad a la hora de desarrollarlo.
2. **Diseño**. Una vez hecha la planificación, hemos diseñado la estructura de esta. En este punto, hemos hecho planos con las conexiones entre los componentes de la aplicación y bocetos de los diseños de las diferentes pantallas.
3. **Implementación**. Una vez están sentadas las bases, hemos pasado a desarrollar la aplicación. Partimos desde la parte visual y, en primer lugar, hemos desarrollado la navegación entre pantallas. A partir de aquí, se fueron añadiendo secuencialmente las diferentes funciones que componen la aplicación.
4. **Verificación**. Tras terminar el desarrollo, realizamos una serie de pruebas en las que buscábamos posibles lagunas y errores en el funcionamiento. En esta fase también tendríamos la reparación de dichos errores encontrados.
5. **Mantenimiento**. Es la fase final en la que la aplicación está terminada y se entrega. A partir de aquí, tenemos que darle mantenimiento y mejorarla con nuevas funciones.

Antes de empezar, barajamos también aplicar metodologías ágiles. En concreto, el método de desarrollo ágil llamado **Scrum**[7]. Sin embargo, está más pensado para desarrollar aplicaciones de forma colaborativa, por lo que no era el más conveniente.

En este caso, la decisión más lógica era la metodología en cascada. Así, hemos podido desarrollar la aplicación en una línea secuencial conforme alcanzábamos los diferentes hitos establecidos.

El proyecto ha sido totalmente desarrollado utilizando Visual Studio Code, un editor de código fuente desarrollado por Microsoft que proporciona muchas facilidades, entre las que destaca su buena integración con Git[8].

Existen cientos de extensiones para VS Code y siempre se están desarrollando más. Para ayudarnos con el desarrollo de esta aplicación, hemos utilizado las siguientes:

- **GitLens** ayuda a visualizar el historial de cambios, entre otras funciones.
- **Vetur** proporciona una caja de herramientas para programar con Vue (remarcado de sintaxis, formateado del código, autocompletado, etc.).
- **npm** permite instalar dependencias en la aplicación.
- **Python** ofrece soporte para el lenguaje Python (formato del código, opciones de depuración y más funciones que permiten trabajar cómodamente con dicho lenguaje).

La aplicación consta, principalmente, de tres partes fundamentales:

El **frontend** es la parte del sitio web con la que interactúan los usuarios, razón por la que decimos que es la parte que está del lado del cliente[9]. Hemos desarrollado esta parte creando un proyecto de **VueJS**, un framework progresivo de **JavaScript** [10] de código abierto para la construcción de interfaces de usuario y aplicaciones SPA (Single-Page Application)[11]. Se trata de uno de los frameworks más utilizados actualmente para desarrollar aplicaciones web. En dicho proyecto, hemos creado una serie de componentes utilizando para ello etiquetas de **HTML**, que es un lenguaje de marcado que nos permite definir la estructura del contenido web[10][12]. Para estilizar dichos componentes se ha usado lenguaje **CSS**, que permite añadir un diseño visual a las interfaces de usuario escritas en HTML[10][13].

El **backend** es la parte que está del lado del servidor y contiene la lógica de negocio para que el sitio web funcione correctamente[9]. Hemos desarrollado esta parte en un fichero de **Python**, un lenguaje de programación de alto nivel que permite desarrollar aplicaciones de todo tipo[14]. Para realizar la comunicación con la base de datos, hemos usado **Flask**, un micromarco de Python que se utiliza para desarrollar aplicaciones web de forma ágil[15].

Por último, para la **base de datos** hemos utilizado **MariaDB**, un fork (bifurcación) de los desarrolladores de MySQL[16]. Esta base de datos se encuentra montada en un servidor de Apache[17].

Cabe también comentar que hemos registrado todo el proceso de desarrollo haciendo uso de **Git**, un sistema de control de versiones[18]. Esto nos permite poder volver a versiones anteriores de la aplicación en caso de incompatibilidades o de haber introducido errores en el código.

5.2.- Estructura de la aplicación

Como hemos mencionado anteriormente, la aplicación consta de un proyecto de VueJS donde se encuentra desarrollado el frontend, un script de Python donde se encuentra desarrollada la lógica del negocio de la parte del servidor, y una base de datos montada en un servidor web.

5.2.1.- Estructura del frontend

En el **frontend** tenemos principalmente los elementos visuales de la aplicación. A continuación, podemos ver la estructura de las carpetas que lo forman.

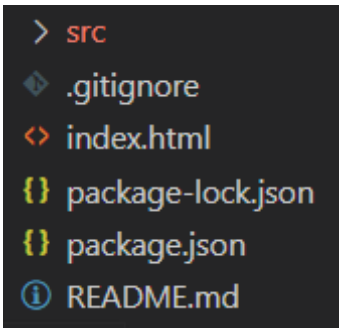

Ilustración 13: Estructura de carpetas del frontend

Vamos a dejar la carpeta **src** para más adelante y vamos a empezar por hablar sobre los distintos ficheros que contiene la carpeta raíz del proyecto y cuáles son sus funciones:

- En el fichero **.gitignore** le podemos indicar a Git qué ficheros queremos que ignore.
- El fichero **index.html** contiene configuración estática del sitio web que no será procesada por el framework. En este fichero, por ejemplo, establecemos el título que se muestra en la pestaña del navegador.
- En el fichero **package-lock.json** encontramos un histórico de versionado de apoyo para el fichero package.json.
- En el fichero **package.json** tenemos la configuración del proyecto. Aquí podemos establecer versiones a nuestra aplicación y se listarán las dependencias que hemos instalado utilizando npm.
- El archivo **README.md** suele utilizarse para documentar información sobre el proyecto.

Pasemos ahora a lo importante, ya que en la carpeta **src** es donde tenemos realmente los ficheros que forman el proyecto. La estructura de esta carpeta es la siguiente:

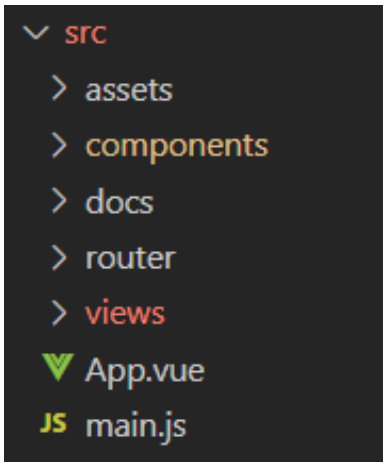

Ilustración 14: Carpeta src

Vamos a comentar su contenido detenidamente:

En el directorio **assets**, se encuentran todos los ficheros utilizados para adornar el sitio web: los iconos utilizados para los botones, el logo, las fuentes empleadas para los títulos y los estilos globales de la aplicación.

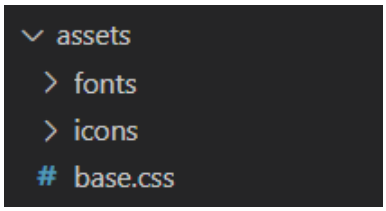

Ilustración 15: Carpeta "assets"

En la carpeta **components**, tenemos los componentes desarrollados para formar parte de las diferentes vistas de la aplicación.

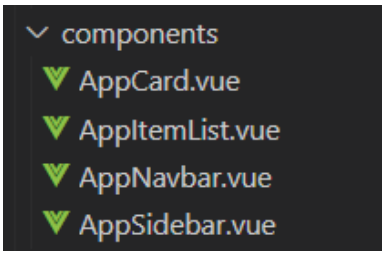

Ilustración 16: Carpeta "components"

- El componente **AppCard** es una tarjeta que se reutiliza en casi todas las vistas para mostrar las películas y las series de forma organizada. En ella, se inyecta el título de las películas o series, la portada, y también el estado, el cual aparece como un resaltado azul añadido a uno de los dos botones que también forman parte de la tarjeta. Veamos un ejemplo de este componente:

Ilustración 17: Componente AppCard con datos de una película inyectados

- El componente **AppItemList** es donde tenemos montados una serie de carruseles de tarjetas para mostrar en la pantalla principal de la aplicación. Estos carruseles se van a ir rellenando cada vez que se acceda a la pantalla principal con las series y películas que el usuario haya marcado como “pendientes de ver”. Así como las series y películas más populares del momento, recogidas directamente de la API de IMDB. Así luce el carrusel que nos muestra las series de televisión más populares del momento:

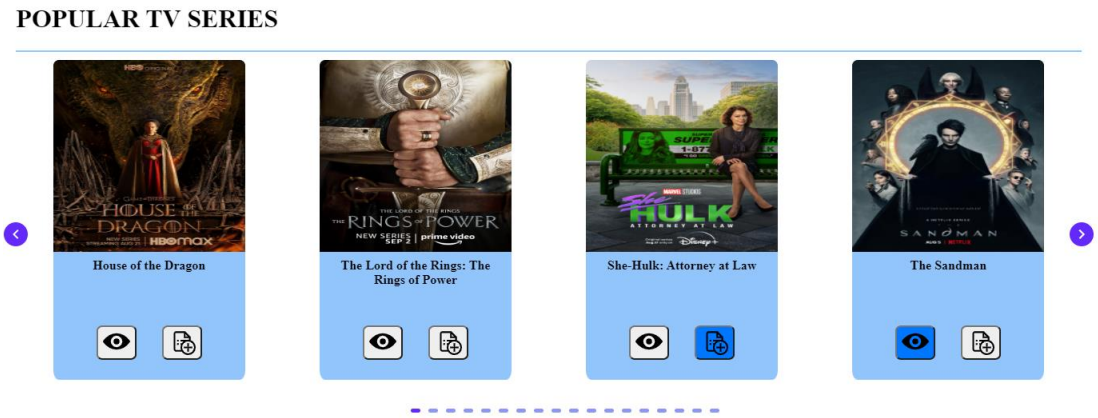

Ilustración 18: Carrusel de series de TV populares

- El componente **AppNavbar** es una barra de navegación fijada a la parte superior de la pantalla durante toda la navegación por la aplicación (salvo en la pantalla de autenticación). En ella, disponemos de los botones que utilizaremos para navegar por las diferentes pantallas.
- El componente **AppSidebar** es un panel lateral que se está utilizando en la pantalla de perfil de usuario para filtrar y ordenar el contenido.

Ilustración 19: Componente AppSidebar

Pasamos ahora a la siguiente carpeta. En la carpeta **docs**, tenemos almacenados algunos manuales de uso (documentaciones) de las diferentes tecnologías utilizadas, así como anotaciones hechas a lo largo del desarrollo de la aplicación.

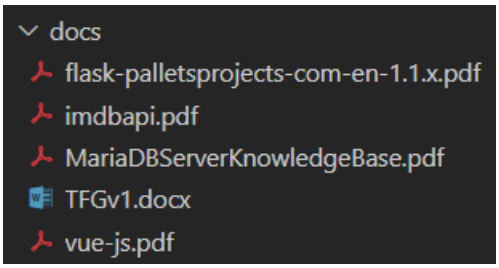

Ilustración 20: Carpeta "docs"

En la carpeta **router** tenemos un fichero de JavaScript que tiene la función de permitir la navegación entre rutas, es decir, entre las diferentes pantallas. En él hemos definido las rutas por las que el usuario va a poder navegar en la aplicación: la pantalla de autenticación (/auth), la pantalla principal (/home), la pantalla del perfil (/profile) y la pantalla de búsqueda (/search). Además, en este fichero también tenemos una función que redirigirá siempre a la pantalla de autenticación a todos aquellos usuarios que no hayan iniciado sesión, impidiendo así el acceso a usuarios no identificados.

Por último, en la carpeta **views**, tenemos las diferentes vistas de la aplicación. Esto es el conjunto de componentes que verá el usuario en cada pantalla: la vista de autenticación (para registrarse e iniciar sesión), la vista principal (que se muestra al iniciar sesión), la vista de perfil (que muestra la colección del usuario) y la vista de búsqueda (donde el usuario puede buscar series y películas).

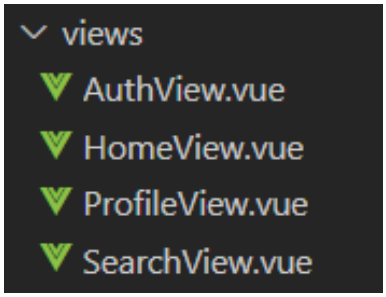

Ilustración 21: Carpeta "views"

Estas vistas no dejan de ser ficheros .vue, al igual que los componentes, pero las usamos como componentes “padre” en los que hemos ido utilizando los componentes “hijo”. Vamos a verlas con más detenimiento:

En primer lugar, tenemos la vista **AuthView**, en la cual el usuario tendrá que iniciar sesión o registrarse para acceder a la aplicación. Esta vista se compone, principalmente, de una imagen a modo de decoración, dos cuadros de entrada de texto en los que el usuario tendrá que introducir un nombre de usuario y una contraseña, y dos botones, uno para iniciar sesión y otro para registrarse, respectivamente.

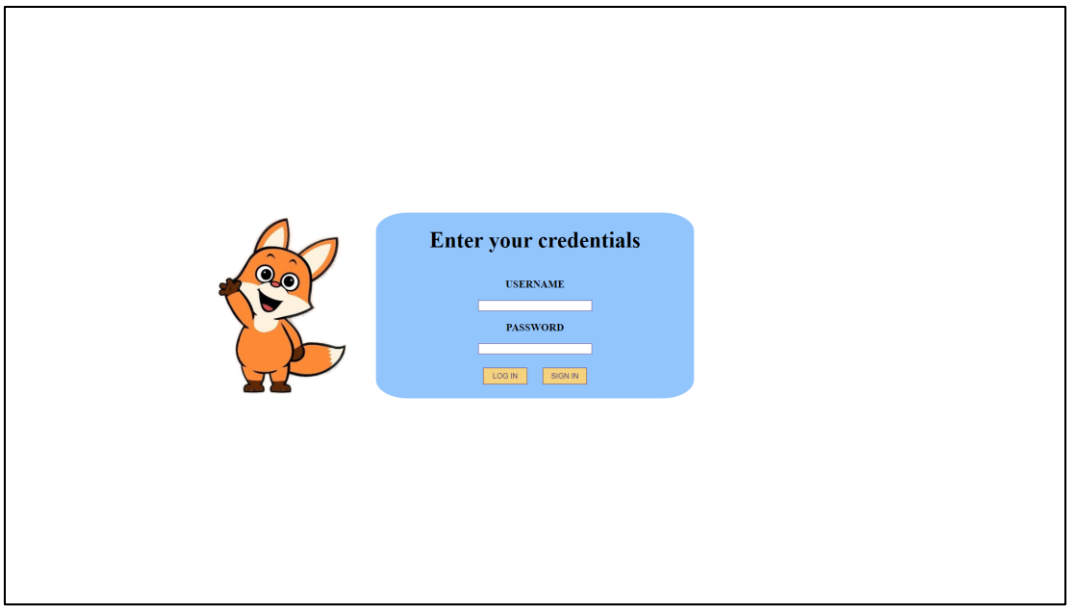

Ilustración 22: Vista de autenticación de usuarios

Además, tiene cuatro mensajes ocultos destinados a actuar de feedback para los datos introducidos por el usuario. Es decir, tenemos dos mensajes para el inicio de sesión: uno para cuando el usuario introduzca sus datos e inicie sesión correctamente, y otro para cuando el usuario introduzca datos incorrectos y no se pueda iniciar sesión.

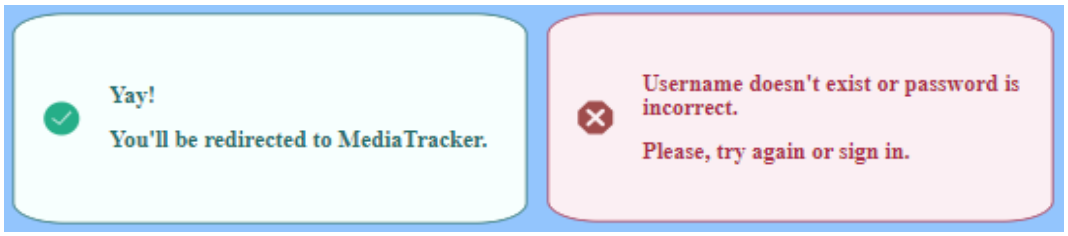

Ilustración 23: Mensajes de inicio de sesión correcto e incorrecto, respectivamente

De igual forma, si el usuario trata de registrarse, también hay dos mensajes para indicar si el usuario ha introducido un nombre de usuario que ya existe o, de lo contrario, se ha podido registrar correctamente.

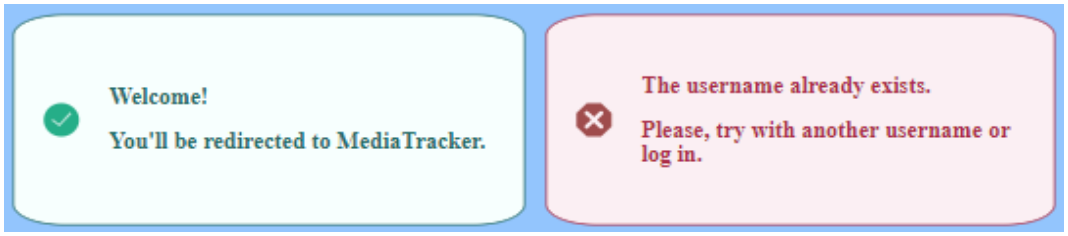

Ilustración 24: Mensajes de registro correcto e incorrecto, respectivamente

Cuando pulsamos los botones de inicio de sesión o registro, el controlador envía una petición al servidor que, en función del resultado de gestionar dicha petición, nos responde con un mensaje de error o de éxito. El controlador gestiona dicho mensaje y actualizará la vista añadiendo uno de los mensajes de las ilustraciones anteriores.

A continuación, tenemos la vista HomeView, que se corresponde con la pantalla principal de la aplicación. Se trata de la vista a la que vamos a redirigir automáticamente al usuario cuando inicie sesión. En ella, se ha utilizado principalmente el componente AppItemList, mencionado con anterioridad.

En primer lugar, se muestran dos carruseles con las películas y series, respectivamente, que el usuario en cuestión quiere ver. Después tenemos otros dos carruseles con películas y series populares en la actualidad. Por último, tenemos otro carrusel con películas que se encuentran ahora mismo en cines.

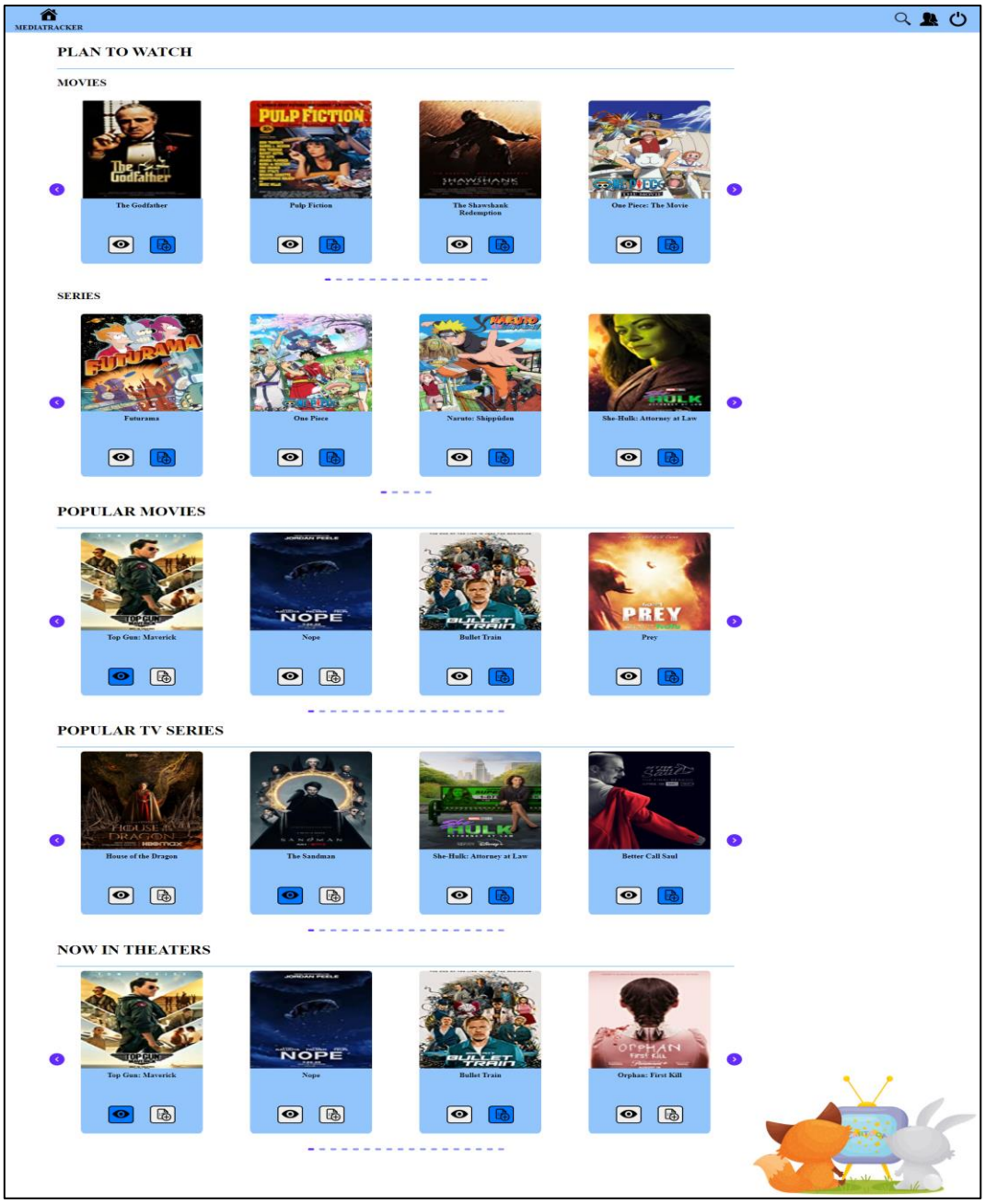

Ilustración 25: Vista principal de la aplicación

Hemos utilizado aquí el ciclo de vida de los componentes de VueJS[19]. Concretamente, hemos utilizado el método **mounted**. De esta manera, cada vez que accedamos a la pantalla principal, se realizarán cinco peticiones (una para cada carrusel) con el objetivo de que el usuario ya vea el contenido cargado nada más acceder.

Para los dos primeros carruseles, pediremos a nuestro modelo que nos envíe todo el contenido que el usuario ha añadido a su colección. Una vez que nos llega la respuesta, filtraremos ese contenido, quedándonos únicamente con aquellos elementos que estén marcados como "pendiente de ver".

Para los tres últimos carruseles, hacemos las peticiones a la API de IMDB. La API dispone de varias URLs a las que se les puede pedir contenido. En este caso, hemos decidido que lo más interesante sería mostrarle al usuario el contenido popular y en cines en la actualidad. Cabe

mencionar también que, cuando recibimos el contenido de la API, también hacemos una petición a nuestro modelo. En ella, comprobamos si alguno de los elementos que nos ha enviado la API ya se encuentra almacenado en la colección del usuario. En caso afirmativo, el modelo nos enviará el estado con el que se encuentra almacenado dicho elemento para que podamos resaltarlo en su correspondiente tarjeta y así el usuario sepa que ya lo tiene añadido.

Otra de las vistas es la página de perfil del usuario (**ProfileView**). En ella, se muestra todo el contenido que el usuario ha añadido a su base de datos. Consta de un panel lateral donde hemos añadido opciones de filtro y ordenación y, por supuesto, un espacio en el que van a aparecer las tarjetas con el contenido almacenado en la base de datos del usuario.

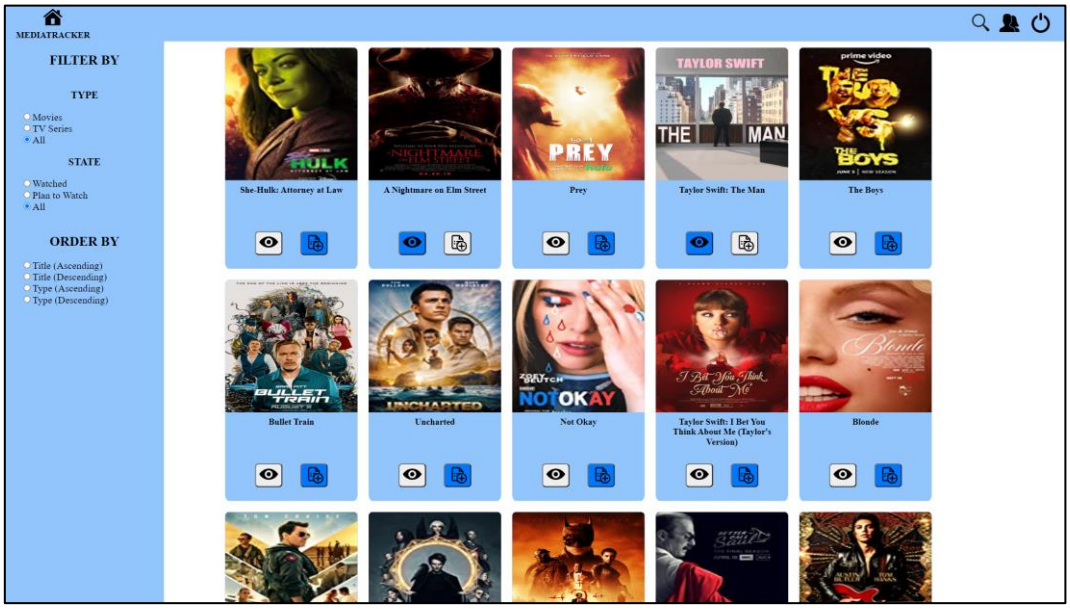

Ilustración 26: Pantalla de perfil de usuario

Al igual que en la pantalla principal, aquí también utilizamos el método **mounted** del ciclo de vida de VueJS para que los datos se carguen automáticamente cuando accedemos a la página. El proceso es igual que en la pantalla anterior: nada más acceder a la pantalla, hacemos una petición al modelo para que nos envíe todo el contenido y, una vez recibido, lo inyectamos en las tarjetas.

Por último, tenemos la pantalla de búsqueda (**SearchView**), en la cual utilizamos un cuadro de entrada de texto para que el usuario escriba en él lo que quiere buscar. Al pulsar la tecla Enter o hacer clic en el botón de buscar, se ejecutan dos peticiones hacia la API de IMDB (una para las series y otra para las películas), que nos enviará un objeto con los resultados que coincidan con la búsqueda. En este punto, volvemos a pedir al modelo que, en caso de que alguno de los elementos que nos llega de la API esté ya en la base de datos, nos envíe su estado para poder resaltar el botón correspondiente de la tarjeta. Finalmente, los resultados de la búsqueda se muestran en pantalla de la siguiente manera:

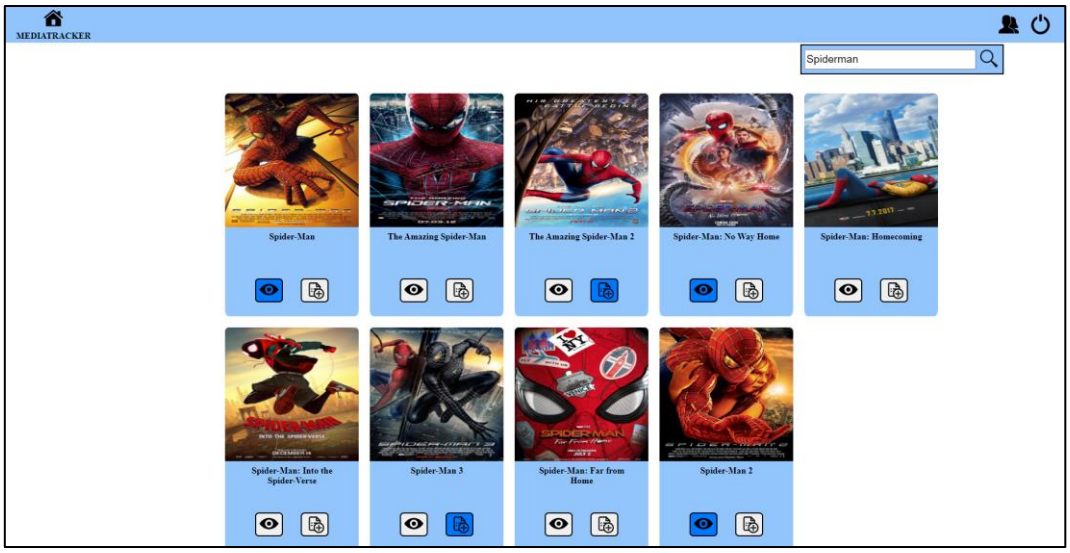

Ilustración 27: Pantalla de búsqueda

5.2.2.- Desarrollo del backend

Como se ha mencionado a lo largo del documento, en el backend tenemos implementada la lógica del negocio. Consiste básicamente en una serie de métodos que van a ejecutarse bajo las peticiones del cliente.

El primer método que tenemos está dirigido al registro de usuarios. En este método vamos a recibir una petición junto con el nombre de usuario y la contraseña introducidos por el usuario. Al ejecutarse este método, el servidor accederá a la base de datos para asegurarse de que el nombre de usuario recibido no existe ya. En ese caso, el nombre de usuario y la contraseña se introducirán en la base de datos de usuarios y automáticamente iniciaremos la sesión del nuevo usuario, enviando al cliente un mensaje de éxito junto con el identificador numérico del usuario para poder mantener activa la sesión de usuario en el navegador. Si, por el contrario, el usuario ya existe, no se podrá registrar, por lo que enviaremos al cliente un mensaje de error indicando la situación.

De la misma forma, tenemos otro método para iniciar sesión que es muy similar al anterior. La única diferencia es que, esta vez, comprobaremos primero que el nombre de usuario existe. Tras verificarlo, comprobaremos que la contraseña introducida por el usuario se corresponde con la contraseña que tenemos almacenada en la base de datos. Si todo esto se cumple, la sesión se iniciará, enviando al cliente un mensaje de éxito junto con el identificador numérico del usuario para mantener la sesión activa en el navegador. En caso contrario, enviaremos un mensaje de error.

Tenemos otros dos métodos que vamos a comentar de forma paralela. Se trata de métodos para añadir un elemento (ya sea película o serie) a la base de datos según su estado, que puede ser “visto” o “pendiente de ver”. En ambos casos recibimos una petición para añadir el elemento a la base de datos. En dicha petición nos vendrá el identificador del usuario que la está añadiendo, el identificador de la serie o la película, su título, su portada y el tipo de contenido que es. En primer lugar, se comprueba si el usuario ya había añadido antes el elemento a la base de datos. En caso afirmativo, entenderemos que el usuario quiere eliminar el elemento de su colección, por lo que esta será la acción realizada. Si el elemento ya existe en la base de datos, pero el estado no se corresponde con el que nos acaba de llegar en la petición (por ejemplo, el usuario quiere marcarlo como visto pero ya lo tenemos guardado como pendiente de ver),

entenderemos que el usuario quiere cambiar el estado del elemento, así que modificaremos la base de datos para actualizarlo. Por último, si el elemento no existe en la base de datos, se añadirá con los atributos recibidos.

El siguiente método por comentar será el que se encargue de enviar a la vista todo el contenido de la base de datos de un usuario. Este método será el responsable de que el usuario pueda ver en su página de perfil todo el contenido que ha ido añadiendo durante su uso de la aplicación. Recibiremos una petición con el identificador numérico del usuario y consultaremos la base de datos, recorriendo la tabla de contenido para buscar todas las filas en las que aparezca dicho identificador. Cada vez que encontremos una fila que contenga el identificador, la añadiremos a una lista. Cuando terminamos de recorrer la tabla, enviamos dicha lista al cliente.

Por último, hemos tenido que añadir un método que nos indique el estado de los elementos, ya que cuando hacemos peticiones a la API de IMDB, el estado no va a verse reflejado en los resultados obtenidos. En este caso, recibimos una lista con los identificadores de los elementos que nos llegan de la API. Lo que hacemos es recorrer la tabla de contenido, recopilando en un diccionario los elementos que ya existan en nuestra base de datos junto con el estado con el que los añadió el usuario en cuestión. Por último, enviamos dicho diccionario al cliente.

5.2.3.- Base de datos

La base de datos se trata de una base de datos relacional simple, apta en este caso ya que hemos desarrollado una aplicación pequeña que, en principio, no tendrá un gran tráfico de usuarios. Hemos seguido las pautas que diseñamos en el capítulo 4 del documento, pero se han introducido una serie de modificaciones.

Nuestra base de datos está formada por dos tablas relacionales entre sí: una para registrar usuarios y la otra para almacenar el contenido añadido por todos ellos. Esto deberá cambiar si la aplicación llega a tener un tráfico relativamente alto de usuarios, lo cual permitiría realizar más funciones.

Hemos llamado a la primera tabla con el nombre de **user**. En esta tabla tenemos tres columnas para almacenar datos referentes al registro de los usuarios: sus nombres de usuario y contraseñas, que serán datos de texto, y un identificador numérico auto incremental que será único para cada usuario.

Por otro lado, tenemos una tabla para el contenido, a la que hemos llamado **content**. En esta tabla iremos añadiendo las series y películas que indiquen los usuarios. Disponemos de seis columnas: las cuatro mencionadas en el diseño (identificador de la serie o película proveniente de la API de IMDB, tipo de elemento, estado marcado por el usuario e identificador del usuario que lo ha añadido) y, además, también iremos añadiendo los títulos y las portadas de las series y películas almacenadas. De esta manera podremos, con el tiempo, no depender tanto de la API de IMDB.

5.3.- Problemas de implementación resueltos

5.3.1.- CORS

El **CORS** (Cross Origin Resource Sharing) es un mecanismo basado en cabeceras HTTP que permite a los servidores indicar a los navegadores si deben permitir la carga de recursos para un origen distinto al suyo[20].

Este mecanismo nos impedía realizar conexiones entre el cliente y el servidor, ya que nuestras peticiones, tanto las realizadas a nuestra base de datos como las realizadas a la API de IMDB, consistían en utilizar una llamada fetch().

Este método usa CORS y, cuando se detecta que el navegador está cargando recursos de un origen distinto al suyo, nos aparece en la consola del propio navegador el siguiente mensaje:

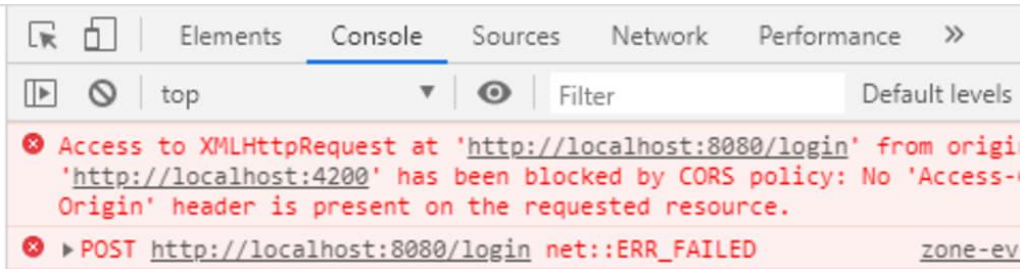

Ilustración 28: CORS

La solución aplicada ha consistido en sustituir las llamadas en las que utilizábamos el método fetch() por una biblioteca de JavaScript llamada Axios.

Axios dispone de un método get() que realiza la misma función que el método fetch(), con la ventaja de que, además, transforma automáticamente los datos recibidos del servidor en formato JSON, lo cual nos facilita trabajar con ellos.

De igual forma, en caso de querer utilizar fetch() o de que más adelante Axios diera el mismo problema, se pueden añadir al código unas configuraciones para modo de las solicitudes, siendo una de ellas la de prevenir el CORS[21].

6.- Manuales

6.1.- Guía de ejecución y mantenimiento

Para ejecutar este proyecto en local y poder darle mantenimiento, necesitamos una serie de herramientas y seguir ciertos pasos.

Lo primero será descargar e instalar XAMPP, un entorno de desarrollo distribuido por Apache que incluye varios softwares libres. Podemos descargarlo desde la página web oficial de Apache Friends[22].

Una vez lo tenemos instalado, lo abrimos y ejecutamos Apache y MySQL haciendo clic en los botones de Start correspondientes.

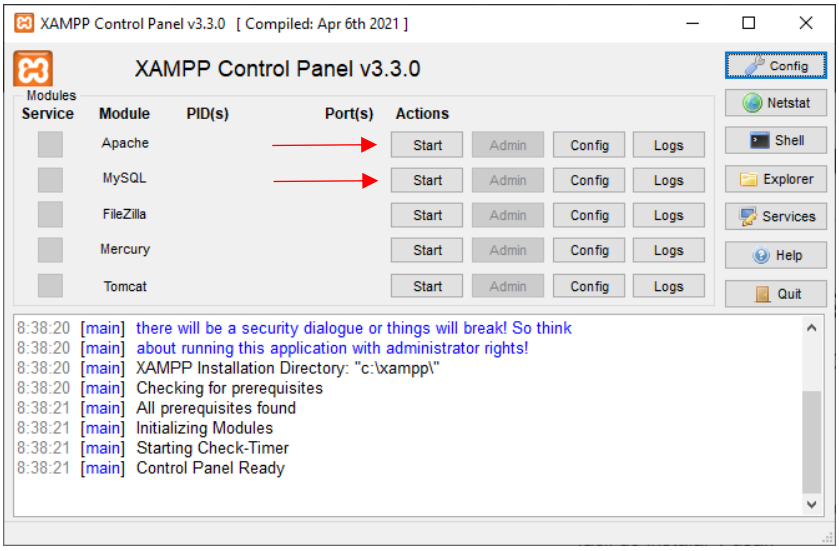

Ilustración 29: XAMPP Control Panel

Con esto, tendremos el servidor de Apache escuchando, generalmente, en el puerto 80 (podemos configurar otro puerto si queremos pulsando el botón Config) y la base de datos de MySQL (MariaDB, realmente) instalada en dicho servidor.

Si accedemos a la dirección localhost:80 veremos la pantalla de bienvenida de Apache y, en la parte derecha de la barra de navegación superior, veremos un botón en el que se puede leer phpMyAdmin. Esta es una herramienta que, si bien no es estrictamente necesaria, nos sirve de interfaz para poder administrar la base de datos de forma sencilla y gráfica: añadir tablas, añadir columnas a dichas tablas o incluso crear nuevas bases de datos.

Ilustración 30: Página de bienvenida de Apache

Una vez hemos puesto en marcha el servidor, tenemos que descargar algún editor de código fuente para abrir el proyecto. Para desarrollar la aplicación hemos utilizado Visual Studio Code, por lo que se recomienda también su uso. Se puede descargar en un página web oficial[23].

Al abrir el proyecto con dicho programa, necesitaremos instalar las dependencias. Para ello, abrimos una terminal (pulsamos Ctrl+J o hacemos clic en View->Terminal) y tecleamos el comando **npm install**. Quizá tengamos que instalar NodeJS y npm en nuestro ordenador para poder ejecutar este comando[24].

Una vez instaladas las dependencias, ejecutamos en la terminal el comando **npm run dev**, lo cual ejecutará el frontend en el puerto 3000 (podemos cambiarlo en los archivos de configuración del proyecto).

Por último, desde una consola (o incluso abriendo una nueva terminal en VS Code), nos desplazamos hacia el directorio en el que tenemos almacenado el script de Python que contiene la lógica de negocio y ejecutamos el comando **python.exe flaskserver.py** (python.exe seguido del nombre del fichero de Python, incluyendo su extensión).

En este punto ya tendremos nuestra aplicación ejecutándose y podremos navegar por ella o modificar su código fuente en VS Code para introducir mejoras y/o reparar errores.

6.2.- Guía de uso

Para utilizar esta aplicación, hay que empezar por crear una cuenta de usuario. Para ello, introducimos un nombre de usuario y una contraseña en los cuadros de texto de la pantalla de inicio y pulsamos el botón de “SIGN IN”. En caso de tener ya un usuario registrado, solo habrá que rellenar las credenciales y pulsar el botón de “LOG IN” para iniciar sesión. En cualquier caso, tras pulsar el botón, la aplicación nos redirigirá a la pantalla principal.

Ilustración 31: Pantalla de autenticación

En la pantalla principal, veremos nuestro contenido pendiente de ver, separado en dos carruseles, uno de películas y otro de series. Seguido de esto, veremos las películas y series populares en la actualidad y, más abajo, las películas que se encuentran actualmente en cartelera.

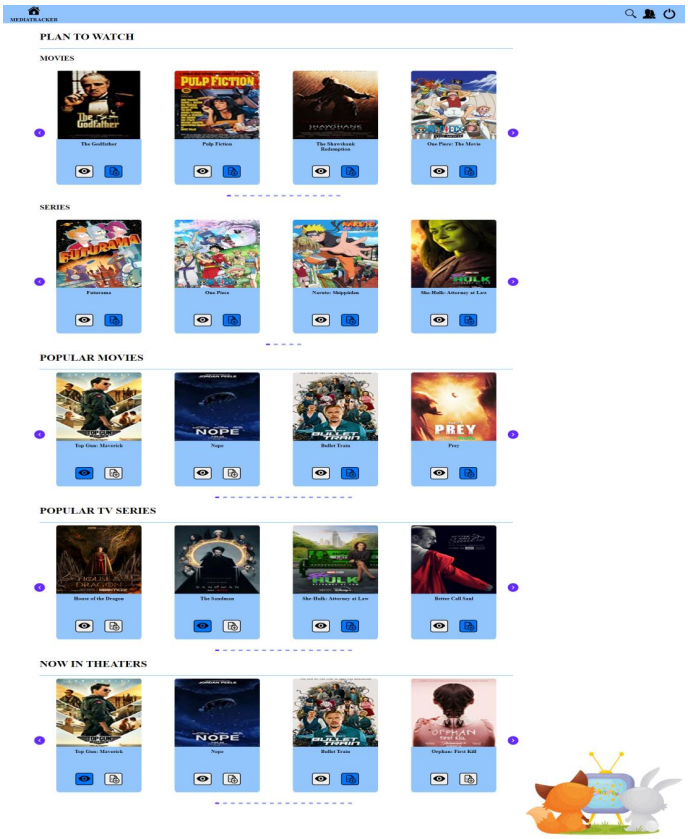

Ilustración 32: Pantalla principal

Podemos añadir a nuestra colección cualquiera de los elementos mostrados pulsando el botón de “WATCHED” (señalizado con el icono 👁) para elementos que ya hayamos visto, o el botón de “PLAN TO WATCH” (señalizado con el icono 📄⊕) para elementos que queramos ver.

Ilustración 33: Tarjeta con vista de película

Si queremos buscar un elemento cualquiera, pulsaremos en el botón de “SEARCH” 🔍 de la barra superior de la pantalla, el cual nos llevará a la pantalla de búsqueda.

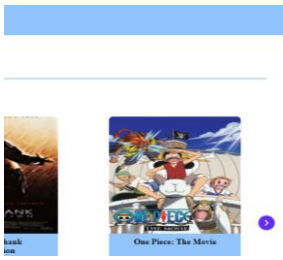

Ilustración 34: Botón para ir a buscar contenido

En ella, escribiremos el nombre de la película o serie que queremos buscar y pulsamos la tecla *Enter* o hacemos clic en el icono de la lupa. A continuación, se mostrarán los resultados obtenidos y podremos añadir lo que queramos a nuestra colección.

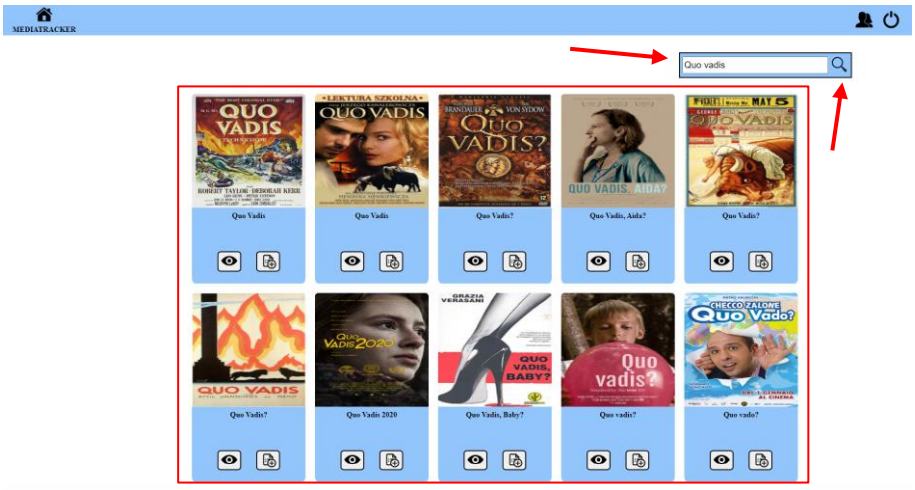

Ilustración 35: Pantalla de búsqueda

Cuando queramos ver todo nuestro contenido, haremos clic en el botón de “PROFILE” .

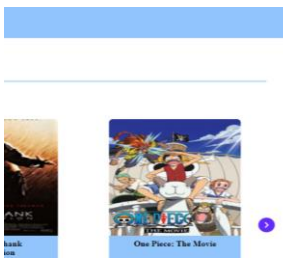

Ilustración 36: Botón para ir al perfil de usuario

Aquí veremos todo nuestro contenido y tendremos la posibilidad de ordenarlo por título o tipo, así como filtrarlo según el tipo de contenido que queramos ver (series o películas) y el estado de estas (vistas o pendientes de ver).

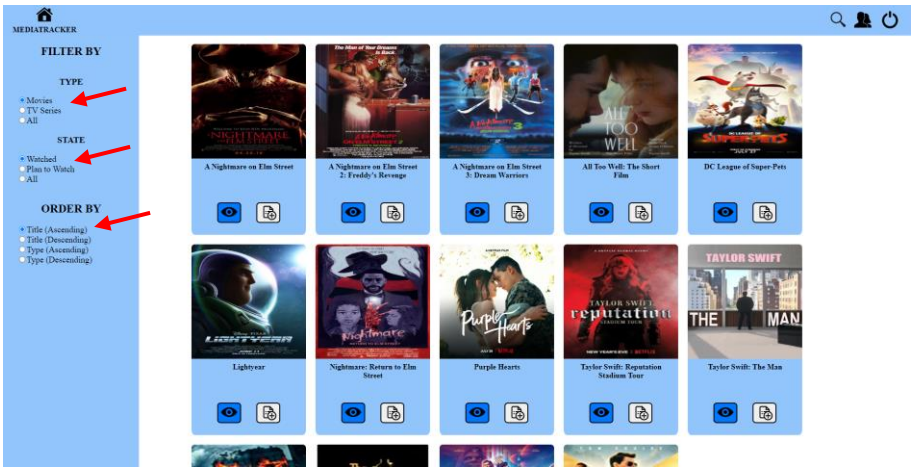

Ilustración 37: Pantalla de perfil

Por último, para salir de la aplicación, haremos clic en el botón de "LOG OUT" ⏻, lo cual cerrará nuestra sesión y nos llevará a la ventana de inicio de sesión.

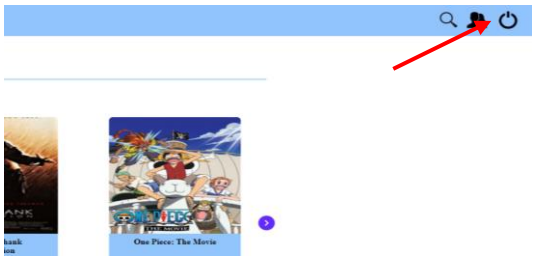

Ilustración 38: Botón para cerrar sesión

7.- Evaluación

Para evaluar nuestra aplicación, nos hemos basado principalmente en su usabilidad, ya que la idea era desarrollar una aplicación sencilla e intuitiva, que no precise de un tiempo de adaptación a la misma, sino que cualquier persona la pueda utilizar cómodamente desde su primer contacto con ella.

Hemos tratado de implementar algunas de las funciones más atractivas de las diferentes plataformas comentadas en el capítulo 2 de este documento, por lo que veo conveniente comparar nuestra aplicación con el resto de las aplicaciones.

En primer lugar, hemos querido basar la pantalla de perfil en la que utiliza Tviso (ver Ilustración 1), habiendo conseguido un buen resultado en comparación, ya que no solo somos capaces de ver en esta pantalla todo el contenido que ha añadido el usuario, sino que además podemos filtrarlo y ordenarlo a nuestro gusto.

Por otro lado, hemos querido ofrecer al usuario su contenido pendiente nada más acceder a la aplicación, sin que tenga que navegar hacia ningún directorio. Consideramos que la principal razón de uso de esta aplicación es para anotar contenido que queremos ver y, conforme lo veamos, anotarlo como visto. Así que me parece indispensable que el usuario pueda ver su contenido pendiente con toda la facilidad del mundo.

Como se mencionaba en la plataforma de Palomitacas, también nosotros estamos ofreciendo a los usuarios una recopilación con el contenido más popular del momento. Dicho contenido viene dado directamente de la API de IMDB, que está en constante actualización, así que los usuarios dispondrán de actualizaciones diarias de contenido popular. Además, también mostramos el contenido que se encuentra actualmente en cines, que podría ser también de interés para los usuarios.

El punto débil de la plataforma TV Time era que estaba únicamente dedicada al seguimiento de series de TV. En todo lo demás, pensamos que el destino de nuestra aplicación debería ser ese, pero implementando además el contenido relacionado con el cine. Por el momento, y a diferencia de TV Time, nuestra aplicación no es capaz de llevar un seguimiento de los episodios de las series, pero sin duda debería ser uno de los puntos a desarrollar en el futuro.

Otra desventaja de nuestra aplicación frente al resto es que todas permiten interacción entre usuarios, mientras que la nuestra no. Sin embargo, nuestra aplicación no ha sido concebida como una red social, sino como una aplicación para que los usuarios simplemente puedan organizar su contenido. Esto no significa que, de cara al futuro, no pueda orientarse el funcionamiento de la aplicación a actuar como una red social que permita a los usuarios agregar amigos, ver el contenido que están viendo sus amigos, escribir comentarios y valoraciones o incluso crear listas con su contenido favorito y para realizar sugerencias a otros usuarios.

La última característica por mencionar es la búsqueda de contenido, comparable a cualquiera de las plataformas mencionadas, ya que está directamente conectada con la API de IMDB, que es una enorme base de datos en línea en la que se encuentra almacenada una gran cantidad de información sobre películas, programas de televisión y demás contenido de entretenimiento audiovisual. Además, como hemos mencionado, se encuentra en constante actualización, por lo que pocas veces no encontraremos lo que buscamos.

La parte negativa de esto es que estamos totalmente a merced de IMDB, que a pesar de ser una base de datos que lleva activa desde 1990, nunca se sabe cuándo podría desaparecer o restringir

el acceso a su API. Por esta razón, vemos conveniente que, a medida que el usuario añade contenido a su perfil, también vayamos copiando información sobre el propio contenido en una base de datos, ya que en un futuro podría servirnos para no tener que depender de IMDB.

8.- Conclusiones

Como conclusión, podemos afirmar que hemos cumplido con muchos de los objetivos del proyecto, ya que hemos conseguido desarrollar una aplicación web sencilla e intuitiva que, si bien es cierto que puede mejorarse, permite a los usuarios realizar un seguimiento del contenido audiovisual (series y películas) que hayan consumido o estén interesados en consumir, que era el propósito más básico de su desarrollo.

Los usuarios de esta aplicación pueden buscar contenidos en la API de IMDB, la cual está continuamente actualizándose, por lo que pocas veces no encontrarán lo que busquen.

El contenido que los usuarios añadan a su colección se irá "copiando" en la base de datos que hemos creado, lo cual nos permitirá no depender tanto de la API de IMDB para según qué funciones, limitándonos a consultarla cuando queramos buscar nuevo contenido.

Además, cabe mencionar que con el desarrollo de esta aplicación hemos podido aplicar algunos conocimientos adquiridos en las asignaturas de programación que hemos cursado durante la titulación, como puede ser el sistema de control de versiones Git o los comandos SQL para realizar consultas a bases de datos, entre otras cosas.

Hemos visto en primera persona el funcionamiento de una aplicación cliente-servidor, habiendo desarrollado ambas partes por separado y también habiendo hecho que exista una comunicación entre ellas.

Por último, pero no menos importante, hemos podido aprender a desenvolvernos con herramientas y tecnologías que no hemos podido ver durante la titulación, como son los lenguajes de programación utilizados para desarrollar la aplicación (JavaScript, HTML, CSS, Python) o los distintos frameworks y micromarcos (VueJS, Flask).

8.1.- Trabajo futuro

Como hemos comentado en el anterior punto, la aplicación cumple con su misión, pero es cierto que puede mejorarse. Aquí vamos a mencionar algunas de las ideas que se nos han venido a la mente para darle más funcionalidades y dinamismo.

A medida que la aplicación vaya incrementando su número de usuarios, la primera tarea a realizar en el futuro será la de mejorar y optimizar la base de datos. Actualmente utilizamos únicamente dos tablas porque no se espera un tráfico elevado de usuarios, pero lo óptimo sería disponer de más tablas. Podríamos decir que necesitaríamos una tabla casi por cada tipo de elemento: una tabla individual para cada usuario, otra tabla para las películas, otra para las series, otra para los directores, etc. De esta manera, evitaríamos repetir datos en una tabla y podríamos manejar futuras funciones más fácilmente.

Tras esto, nos deberemos centrar más en las series, ya que ahora mismo no podemos contabilizar los capítulos que han visto los usuarios, lo cual es algo bastante importante que implementar.

El siguiente paso por dar, podría ser el de añadir una vista en la que podamos ver detalles de una serie o película. Es decir, que al hacer clic sobre, por ejemplo, una película, la aplicación nos lleve a una página en la que se nos amplíe la información sobre esta: sinopsis, reparto de actores, equipo técnico, tráiler, etc. Incluso se podría aprovechar esta vista para añadir una sección de

comentarios y orientar la aplicación a ser una red social en la que los usuarios puedan interactuar entre sí.

Durante la investigación de aplicaciones similares, observamos que algunas plataformas permiten al usuario exportar su contenido y descargarlo en un fichero a su ordenador. Podríamos también aprovechar esto y añadir una función que permita procesar dicho fichero y añadir los elementos que contiene directamente a la base de datos del usuario que lo quiera importar.

Como opcional, también se podrían añadir otro tipo de estilos a la aplicación, tratando de darle más dinamismo: alguna animación para la navegación entre pantallas o algún banner que muestre los estrenos de la semana u otro tipo de contenido.

Bibliografía

[1] Servicios OTT: https://es.wikipedia.org/wiki/Servicio_OTT

[2] Anexo: Programación original distribuida por Netflix: https://es.wikipedia.org/wiki/Anexo:Programaci%C3%B3n_original_distribuida_por_Netflix

[3] Seriesly: https://selectra.es/internet-telefono/television-streaming/seriesly

[4] Tviso, el Apple TV español, echa el cierre tras no conseguir su venta a Rakuten: https://www.eleconomista.es/tecnologia/noticias/10002675/07/19/TViso-el-Apple-TV-espanol-echa-el-cierre-tras-no-poder-cerrar-su-venta-a-Rakuten.html

[5] Modelo-Vista-Controlador: https://developer.mozilla.org/es/docs/Glossary/MVC

[6] Método de desarrollo en cascada: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/el-modelo-en-cascada/

[7] Scrum: https://www.atlassian.com/es/agile/scrum

[8] Visual Studio Code: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/

[9] Frontend y Backend: https://rockcontent.com/es/blog/front-end-y-back-end/

[10] Gauchat, Juan Diego: "El gran libro de HTML5, CSS3 y JavaScript". Marcombo, 2017.

[11] VueJS: https://openwebinars.net/blog/que-es-vue-js-y-que-lo-diferencia-de-otros-frameworks/

[12] HTML: https://desarrolloweb.com/home/html

[13] CSS: https://lenguajecss.com/css/

[14] Matthes, Eric: "Curso intensivo de Python (2ª ED.): Introducción práctica a la programación basada en proyectos". Anaya Multimedia, 2021.

[15] Flask: https://www.epitech-it.es/flask-python/

[16] MariaDB: https://www.vozidea.com/que-es-mariadb-y-ventajas-frente-mysql

[17] Apache: https://www.hostinger.es/tutoriales/que-es-apache/

[18] Git: https://git-scm.com/

[19] Ciclo de vida de VueJS: https://codingpotions.com/vue-ciclo-vida

[20] CORS: https://www.enmilocalfunciona.io/entendiendo-cors-y-aplicando-soluciones/

[21] Request.mode: https://developer.mozilla.org/en-US/docs/Web/API/Request/mode

[22] Apache Friends: https://www.apachefriends.org/es/index.html

[23] Sitio oficial de Visual Studio Code: https://code.visualstudio.com/

[24] Instalar NodeJS y NPM: https://www.cursosgis.com/como-instalar-node-js-y-npm-en-4-pasos/

[25] Evolución del número de suscriptores de Netflix: https://es.statista.com/estadisticas/598771/numero-de-suscriptores-netflix-en-streaming-en-todo-el-mundo/