UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una aplicación web social para la gestión de información cinematográfica

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Peralta Huamani, Pablo Kevin

Tutor/a: Valderas Aranda, Pedro José

CURSO ACADÉMICO: 2023/2024

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación web social para la gestión de información cinematográfica

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Peralta Huamani, Pablo Kevin
Tutor: Valderas Aranda, Pedro José

Curso 2023-2024

Resum

Aquest treball té com a objectiu proporcionar una alternativa accessible i amigable per als usuaris que busquen organitzar pel·lícules que han vist o volen veure de manera eficaç, ja que les aplicacions més conegudes actualment que duen a terme aquesta funció són difícils d'usar per a usuaris inexperts. La principal funció d'esta aplicació web serà permetre als usuaris gestionar les seues llistes de pel·lícules, a més de poder guardar valoracions d'aquestes, compartir-les amb altres usuaris i fer tot això de manera intuïtiva i eficient. Esta aplicació constarà d'una part *back end* desenvolupada en *.NET*, un apartat de *front end* implementat en *Vue.js* i una base de dades *SQL*.

Paraules clau: web, red, social, cine, gestió, pel·licules, .net, vue

Resumen

Este trabajo tiene como objetivo proporcionar una alternativa accesible y amigable para los usuarios que buscan organizar películas que han visto o quieren ver de manera eficaz, ya que las aplicaciones más conocidas actualmente que llevan a cabo esta función son difíciles de usar para usuarios inexpertos. La principal función de esta aplicación web será permitir a los usuarios gestionar sus listas de películas, además de poder guardar valoraciones de las mismas, compartirlas con otros usuarios y hacer todo esto de manera intuitiva y eficiente. Esta aplicación constará de una parte *back end* desarrollada en *.NET*, un apartado de *front end* implementado en *Vue.js* y una base de datos *SQL*.

Palabras clave: web, red, social, cine, gestión, películas, .net, vue

Abstract

This work aims to provide an accessible and user-friendly alternative for users who want to efficiently organize movies they have watched or want to watch. Currently, the most well-known applications that perform this function are difficult for inexperienced users to use. The main purpose of this web application will be to allow users to manage their movie lists, save ratings for films, share them with other users, and do all of this in an intuitive and efficient manner. The application will consist of a *back end* developed in *.NET*, a *front end* section implemented in *Vue.js*, and an *SQL* database.

Key words: web; network; social; cinema; management; movies; .net; vue

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

A lo largo de los años, el sector cinematográfico, ha cobrado un mayor interés en la sociedad, impulsado por la digitalización, la globalización y la aparición de plataformas de streaming [1]. En el caso de la digitalización, esta ha ocasionado que la producción y distribución del contenido, sea mucho más accesible y que alcance a una mayor cantidad de público. En cuanto a la globalización, esta ha facilitado la expansión de las producciones cinematográficas más allá de las fronteras nacionales, dando lugar a una mayor diversificación de estilos y contenidos, tanto a la hora de elaborar una película, o un cortometraje, como de consumirla, por parte del usuario.

Además de la digitalización y la globalización, debemos tener en cuenta la aparición de las plataformas de streaming, las cuales han revolucionado la manera en la que el público accede al contenido audiovisual. Estas plataformas ofrecen una amplia biblioteca de películas, series y cortometrajes, que se va actualizando conforme se van estrenando nuevas producciones, permitiendo que los usuarios tengan la capacidad de elegir entre opciones casi infinitas para entretenerse.

1.1 Motivación

Si se tiene en cuenta el creciente número de producciones audiovisuales, que el público puede ver o consumir, es posible darse cuenta de que es necesaria alguna manera de manejar toda esa información, ya sea para tener controlado el contenido que se desea consumir o el que ya se ha consumido.

Para esto ya existen algunas aplicaciones, pero dado que ofrecen excesivas funcionalidades, en la mayoría de ellas se pierde la amigabilidad para con el usuario y se dificulta el acceso a la solución del problema planteado. Es por eso que se ha creído conveniente la creación de una aplicación web de uso sencillo, que permita a los usuarios enlistar las películas que les sean de interés y hacer una valoración de cada una de ellas.

1.2 Objetivos

El principal objetivo de este TFG es desarrollar una aplicación que permita a los usuarios organizar las películas que han visto o quieren ver en un futuro, mediante la creación de diferentes listas que las contengan. De manera más específica se pueden distinguir estos objetivos:

1. Crear un usuario en la aplicación.

2. Crear listas personalizadas de películas.
3. Conocer las listas de películas populares, mejor valoradas, próximos estrenos y las que están actualmente en cartelera.
4. Buscar películas por título, para obtener su información y poder añadirlas a listas.
5. Ver las listas que han creado otros usuarios.
6. Valorar una película.

1.3 Estructura de la memoria

Este documento estará dividido en las siguientes partes:

1. Introducción: En este apartado se expresan las claves principales del TFG, tales como objetivos, motivación y una pequeña introducción.
2. Estado del arte: Se hace un análisis de las aplicaciones existentes que permiten al usuario registrarse, enlistar películas, valorarlas y conocer información sobre ellas.
3. Metodología: En este apartado se explican las metodologías existentes, que son utilizadas para el desarrollo de aplicaciones y la que se ha elegido para llevar a cabo este proyecto.
4. Análisis de requisitos: En este punto se recogen los requisitos que tiene el desarrollo de esta aplicación, para poder elaborar los diferentes casos de uso a los que dan lugar dichos requisitos. Mediante este proceso, se puede valorar el alcance que va a tener la aplicación y las funcionalidades que se van a desarrollar.
5. Análisis conceptual y diseño: En esta fase de desarrollo se concentra el diseño y creación de la base de datos, además del diseño de la interfaz de la aplicación con la que interactuará el usuario.
6. Desarrollo de la solución: En este apartado se explica el proceso de desarrollo de la aplicación, desde la arquitectura de la aplicación y las herramientas utilizadas, hasta la programación en la parte *back end* [2] y en la parte *front end* [3].
7. Producto desarrollado: Se explica el producto final y las funcionalidades utilizadas desde la visión del usuario.
8. Validación: Se realiza una validación heurística de la aplicación.
9. Conclusiones: En este apartado, se tendrán en cuenta las conclusiones resultantes del proceso de elaboración del proyecto.

CAPÍTULO 2
Estado del arte

Para poder comenzar a desarrollar una aplicación de gestión de información cinematográfica, primero se debe hacer un análisis de las opciones que existen a día de hoy que ofrezcan una solución similar a la que se propone en este TFG. En el siguiente apartado se hace un estudio de las aplicaciones existentes, sus utilidades, sus ventajas y los inconvenientes que tienen.

2.1 Análisis de las aplicaciones web actuales de gestión cinematográfica

Existen diversas opciones para llevar a cabo esta tarea, entre las más conocidas se encuentran las siguientes páginas web:

- Letterboxd: La aplicación más conocida de todas, para gestionar las películas que el usuario ha visto, o que quiere ver y además sirve para poder compartir las listas con los demás usuarios y poder hacer comentarios y valoraciones acerca de las películas. Además de las opciones que se ajustan a los objetivos que se han planteado para este proyecto, también tiene opciones de seguir a otros usuarios y poder ver su actividad, listas creadas y películas vistas. También es posible tener información de las últimas noticias sobre contenido cinematográfico.

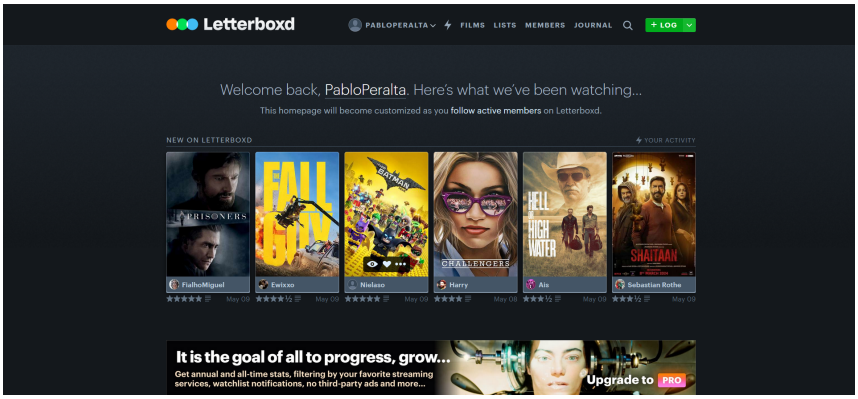

Figura 2.1: Pagina de inicio de Letterboxd

En esta caso se trata de una aplicación muy completa con muchas opciones y funciones muy útiles para los usuarios, pero puede ser difícil de utilizar para personas que no tengan mucha experiencia accediendo a aplicaciones web, por el gran núme-

ro de funcionalidades disponibles, además de estar disponible solamente en inglés, razón por la que no es tan utilizada en España.

- Filmaffinity: Una de las páginas web más utilizadas para conocer información de películas y series de televisión. Se trata de una aplicación que también permite la opción de que los usuarios se registren y creen sus propias listas de películas y/o series, valoren las películas o series si ya las han visto, hagan sus propias críticas sobre ellas y puedan añadir amigos para poder permanecer conectados y compartir contenido. Sin embargo también se trata de una aplicación web que abarca una gran

Figura 2.2: Página de inicio de Filmaffinity

cantidad de funcionalidades que el usuario puede utilizar. Tantas opciones hacen que se pierda el objetivo principal de una aplicación para gestionar el contenido cinematográfico.

- JustWatch: Aplicación web que va creciendo en popularidad, ya que ofrece, no solamente todos los objetivos descritos anteriormente, para la aplicación que se desarrolla en este TFG, sino que también ofrece información acerca de dónde ver el contenido cinematográfico, ya sea por alguna plataforma de streaming, o en cines.

  Además de ofrecer información de contenido cinematográfico, también ofrece información sobre deportes y la emisión de diversas competiciones deportivas.

  Esta aplicación es una buena ejemplificación del proyecto que se quiere realizar, pero tiene algunos inconvenientes. Algunos de ellos son: el desvío del foco principal de este proyecto, con motivo de ofrecer también información acerca de deportes y la posibilidad de crear únicamente dos listas personalizadas de películas o series, si no se obtiene una suscripción de pago.

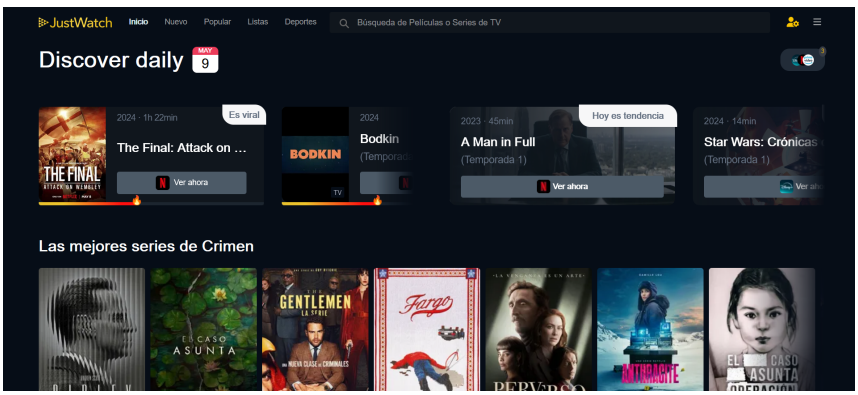

Figura 2.3: Página de inicio de JustWatch

2.2 Críticas al estado del arte

Habiendo hecho ya un análisis de algunas de las opciones de las que dispone un individuo, a la hora de gestionar el contenido cinematográfico que consume o desea consumir, se puede llegar a la conclusión de que las opciones más conocidas actualmente, tienen como objetivos algunos de los presentados anteriormente para este trabajo, pero distan de dar una solución accesible para el usuario y sencilla de manejar para cualquier persona. Al añadir tantas funcionalidades, que pocas veces son utilizadas por un individuo que simplemente quiere llevar un control de las películas que ve o desea ver, ocasiona que estos potenciales usuarios se vean obligados a recurrir a blocs de notas del teléfono móvil, del ordenador o incluso hojas de excel, para ver sus listas de películas de manera intuitiva y sencilla.

CAPÍTULO 3
Metodología

Las metodologías de desarrollo de *software* [4] son conjuntos de técnicas y métodos, que sirven para organizar equipos de trabajo encargados de diseñar soluciones informáticas. El objetivo de estas metodologías es optimizar el proceso de desarrollo, considerando factores como: costes, planificación, dificultad y recursos.

Sin una metodología clara, el desarrollo puede llegar a ser complejo y propenso a problemas, retrasos y errores. Adoptar una metodología correcta es clave para garantizar eficiencia y calidad en la creación de aplicaciones.

3.1 Metodologías de trabajo afines al desarrollo de una aplicación web

Dentro de las metodologías podemos distinguir dos tipos: Las metodologías tradicionales y las metodologías ágiles.

- Metodologías tradicionales: Se caracterizan por una definición integral de los requisitos desde el inicio del proyecto. Se diferencian por ser menos flexibles y permitir menos cambios que las metodologías ágiles.
    - Cascada: Esta metodología, como indica el nombre, organiza las etapas del proyecto de arriba a abajo. Por lo tanto, para que el producto pase de una etapa a la siguiente solo lo puede hacer si ha terminado la etapa en la que estaba. Por esta razón solamente se verá un avance claro del desarrollo cuando sean las etapas finales del proyecto.
    - Prototipado: se basa en la elaboración de un prototipo de manera rápida, para que los usuarios puedan probarlo y dar su opinión y feedback acerca de la funcionalidad, posibles errores en el diseño y la sugerencia de nuevos requerimientos.
    - Espiral: Se trata de una unión de las dos anteriores, añadiendo un análisis de riesgo. Se divide en cuatro pasos: la planificación, el estudio de riesgos, la creación del prototipo y la verificación de los clientes. El funcionamiento es como su nombre indica, en espiral, por lo cual conforme nos acercamos al centro de la espiral mayor será el avance del proyecto.
    - Incremental: En esta metodología de desarrollo, se va construyendo el producto final porogresivamente. Se van aumentando funcionalidades a medida que se avanza en las etapas de desarrollo, esto hace que se vean resultados con mayor rapidez, en comparación a los anteriores modelos. Es mucho más flexible que las demás metodologías tradicioinales.

- Diseño rápido de aplicaciones (*RAD*): Esta metodología permite desarrollar *software* de calidad en poco tiempo. Los costes son altos, el desarrollo flexible y requiere una gran intervención de los usuarios. El código suele tener más errores en este modelo y añade pocas funcionalidades por cada pequeño desarrollo.

- Metodologías ágiles: Estas metodologías vienen claramente influenciadas por el modelo incremental, ya que se basan en agregar pequeñas nuevas funcionalidades a la aplicación, de manera que el producto se acerque a la versión final. Esto permite que los ciclos de desarrollo sean más cortos y se pueda construir equipos de trabajo más pequeños y más independientes.Algunos ejemplos son:

  - *Kanban*: Metodología basada en dividir las tareas en porciones menores y organizarlas en un tablero que distinga entre tareas pendientes, en curso y finalizadas. Esto permite saber de manera visual cual es el avance del proyecto en cualquiera de las fases de desarrollo.
  - *Scrum*: Este modelo también divide las tareas en porciones de desarrollo más pequeñas, pero en este caso existen iteraciones de desarrollo de poco tiempo. Cada iteración se puede llamar también *sprint*. Cada una de las etapas a iterar serían: planificación (*planning sprint*), ejecución (*sprint*), reunión diaria (*daily meeting*) y demostración de resultados (*sprint review*).
  - *Lean* [5]: En esta metodología hay dos objetivos claros: eliminar los desperdicios y poner en valor el respeto de los trabajadores a cualquier nivel. Eliminar los desperdicios significa en este caso, identificar cambios que se pueden realizar en el proceso de desarrollo y adoptarlos, eliminando así despilfarro y aumentando la eficiencia del proceso. En el caso del respeto a los trabajadores a cualquier nivel, también se hace referencia a que se debe reconocer el valor de las ideas de todo el equipo y de los clientes, para tener en cuenta diferentes puntos de vista.
  - Programación extrema [6]: Metodología de desarrollo que se basa en la cohesión del equipo. La clave es conseguir un buen ambiente de trabajo y tener retorno por parte de los clientes, en cuanto a las impresiones, opiniones y pruebas que realizan sobre el proyecto realizado. En esta metodología es muy importante la comunicación, la simplicidad y la retroalimentación, ya que está sujeta a la adaptación a los cambios.

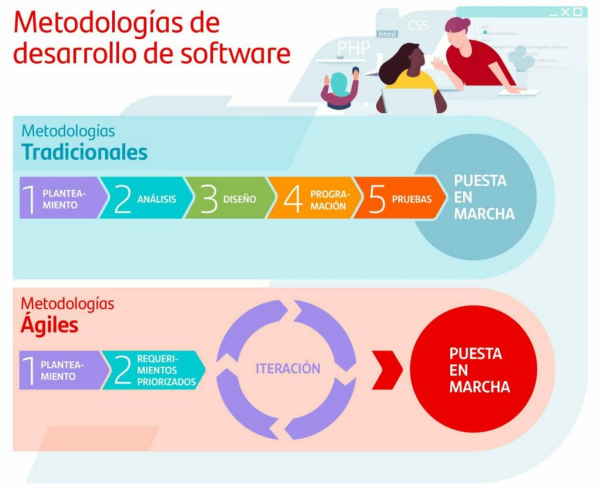

Figura 3.1: Metodologías tradicionales y ágiles

3.2 Metodología adoptada: Incremental

Para la realización de este TFG se ha optado por una metodología incremental [7], ya que es muy flexible, adaptable al cambio y se ajusta al tamaño de este proyecto.

Este método de trabajo, se podría identificar como un ciclo de vida que ocurre durante el desarrollo de *software*, debido a que descompone el proceso completo en diferentes incrementos, que ocasionan un aumento de las funcionalidades del producto. Para cada uno de estos incrementos, también llamados *sprints*, existen cuatro pasos a seguir para que la planificación sea la correcta:

1. Análisis: En este paso, se tienen en cuenta los requisitos del incremento que se desea implementar, para posteriormente realizar un diseño correcto de la solución.
2. Diseño: Una vez se tiene el análisis de la funcionalidad a implementar, se crea un diseño que aborde la implementación de los requisitos analizados.
3. Código: Tras realizar el diseño de la implementación, se desarrolla el código que corresponde a la solución.
4. Pruebas: Una vez se desarrolla la funcionalidad asociada al incremento, se debe probar la solución aplicada, analizar los resultados de las pruebas y validarlas.

Una vez realizada la implementación de un incremento se añade al incremento anterior, con lo cual se van añadiendo funcionalidades al producto, hasta que se completa el desarrollo de toda la aplicación. Esto hace que los clientes puedan ver el avance de la aplicación con cada nueva entrega.

Para esta aplicación, se han realizado diferentes *sprints*. El primer *sprint* consistió en crear las bases del *back end*, que hacen referencia a las entidades, los servicios y los controladores. Con esto ya era posible hacer pruebas CRUD[1], mediante Postman, a las funcionalidades aplicadas. En el segundo *sprint* ya se empezó a crear el *front end* y un esqueleto, de lo que sería a continuación, creando las rutas a las diferentes pantallas que hay en la aplicación y empezando a crear la interfaz con componentes básicos. A continuación se implementó el sistema de inicio de sesión y registro, que involucraba a las dos partes, tanto *back end* como *front end*. Posteriormente a la implementación de la autenticación, se hacía un *sprint* cada vez que se implementaba una pantalla y sus interacciones con base de datos y *API*[2] de TMDB [8]. Por último los *sprints* finales estuvieron enfocados en probar la aplicación y el funcionamiento de toda la estructura desarrollada.

[1] *CRUD*: Acrónimo de *Create* (Crear), *Read* (Leer), *Update* (Actualizar) y *Delete* (Eliminar), representa las acciones básicas de base de datos

[2] *API*: Interfaz de Programación de Aplicaciones, conjunto de funciones y procedimientos que pueden ser reutilizadas en otros sistemas

CAPÍTULO 4
Análisis de requisitos

Para poder comenzar a desarrollar la solución a los objetivos planteados, primeramente se realizará un análisis de los requisitos que se desea abordar. Posteriormente con los requisitos ya definidos, se realizará un estudio de los casos de uso que se deben afrontar en esta solución. Con esto ya se puede realizar un diseño de las interfaces, que también se verá en este capítulo.

4.1 Requisitos iniciales

Para comenzar con el desarrollo de la aplicación, se debe tener claros los requisitos iniciales que ha de cumplir el proyecto que se llevará a cabo. En este caso podemos definir los siguientes requisitos:

- Autenticación de usuarios: Se debe implementar un sistema que permita a los usuarios de la aplicación registrarse, iniciar sesión y cerrar sesión.
- Gestión de listados de películas: El usuario debe poder ser capaz de crear listas de películas con un nombre asociado y la capacidad de eliminar dicha lista cuando sea necesario.
- Búsqueda y adición de películas a listas: Se implementará un buscador de películas, que permita a los usuarios añadir cualquier película a una de sus listas buscándola por el título.
- Películas más relevantes: Se mostrarán listados de películas en la pantalla de inicio, que corresponderán a las películas más populares actualmente, películas actualmente en cines, próximos estrenos y las películas mejor valoradas.
- Listas de la comunidad: Se podrán ver las listas que hayan creado otros usuarios.
- Información de la película: Una vez seleccionada una película, el usuario podrá ver información relevante de la misma.
- Valoración de películas: Un usuario tendrá la capacidad de valorar cualquier película del cero al diez.
- Sugerencia de película: Habrá una función que permita al usuario solicitar la sugerencia de una película al azar, para que pueda descubrir un título desconocido y que pueda ser de interés para su visualización.

Con todos los requisitos principales ya identificados se puede pasar a la construcción de los casos de uso de la aplicación.

4.2 Casos de uso

Primeramente se muestran todos los casos de uso que se van a describir más adelante, en un diagrama en el que de manera gráfica se ve como un usuario debe registrarse o iniciar sesión en la aplicación, para acceder a todas las demás funcionalidades que ofrecerá la aplicación.

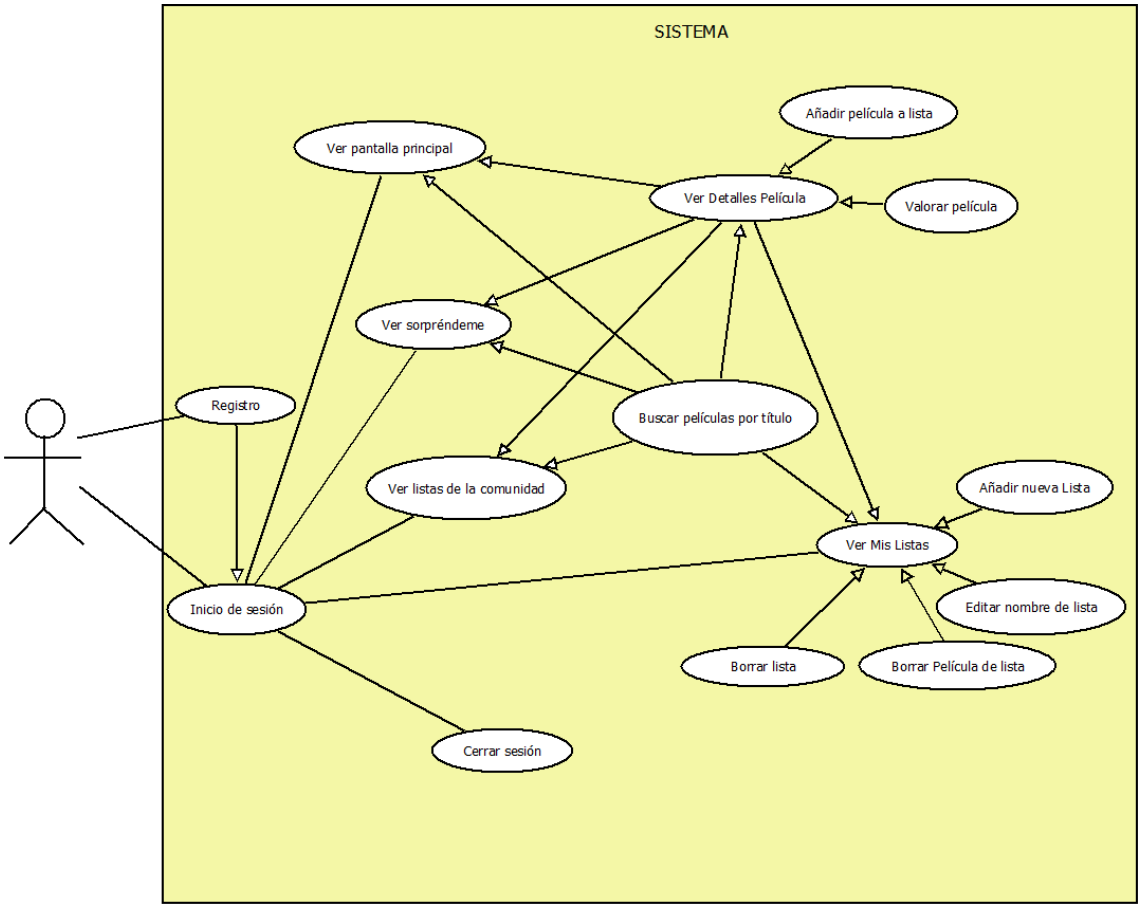

Figura 4.1: Diagrama UML de Casos de uso

A continuación se definen los casos de uso [9] que se tomarán en cuenta para el desarrollo de la aplicación. Siendo estos los siguientes:

- Caso de uso: Registro
  - Descripción: El usuario rellena los campos de registro con los siguientes datos: nombre, apellidos, teléfono, email, nombre de usuario, contraseña y avatar eligiendo una de las dos opciones disponibles.
  - Actor: El usuario sin autenticar.
  - Precondiciones: El usuario no debe estar registrado, y por lo tanto no estaría autenticado en la aplicación.
  - Flujo de acontecimientos:
    1. El usuario pulsa el botón de registrarse en lugar de el botón de inicio de sesión y accede a la pantalla de registro.
    2. El usuario completa los campos y se registra.
  - Postcondiciones: Se comprueba la validez de los datos añadidos y se regresa a la pantalla de inicio de sesión.

- Caso de uso: Inicio de sesión
  - Descripción: El usuario rellena los campos de inicio de sesión, con los valores correspondientes para el nombre de usuario y la contraseña.
  - Actor: Un usuario registrado sin autenticar.
  - Precondiciones: El usuario no debe estar autenticado y debe estar registrado.
  - Flujo de acontecimientos:
    1. El usuario accede a la aplicación web y se le presenta el formulario de inicio de sesión.
    2. El usuario completa los campos y pulsa el botón de inicio de sesión.
  - Postcondiciones: Se comprueba la validez de los valores de los campos introducidos por el usuario. Si son correctos accede a la pantalla principal de la aplicación. Si son erróneos no podrá acceder a la aplicación.
- Caso de uso: Cerrar sesión
  - Descripción: El usuario cierra sesión y dejan de ver el contenido de la aplicación.
  - Actor: Un usuario autenticado.
  - Precondiciones: El usuario debe estar autenticado
  - Flujo de acontecimientos:
    1. El usuario pulsa el botón de cerrar sesión.
    2. Se abre una ventana de confirmación y el usuario confirma la acción y deja de estar autenticado. Si no se confirma la acción, se cancela y se regresa a la vista en la que estaba el usuario.
    3. Se muestra nuevamente la pantalla de inicio de sesión, para poder volver a iniciar sesión
  - Postcondiciones: Ninguna
- Caso de uso: Ver pantalla de inicio
  - Descripción: El usuario ve la pantalla principal de la aplicación que muestra cuatro listas relevantes de películas: Populares actualmente, películas en cartelera, próximos estrenos y mejor valoradas.
  - Actor: Usuario autenticado.
  - Precondiciones: El usuario debe estar autenticado.
  - Flujo de acontecimientos:
    1. El usuario inicia sesión y accede a la pantalla principal de la aplicación, o accede a la pantalla principal mediante el item de menú de la izquierda de la pantalla.
    2. Se cargan las listas de películas que se deben mostrar en la pantalla de inicio.
  - Postcondiciones: Ninguna
- Caso de uso: Buscar películas
  - Descripción: El usuario puede buscar películas por su título.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado.

  - Flujo de acontecimientos:
    1. El usuario escribe un caracter en la barra de búsqueda
  - Postcondiciones: Se muestran los resultados ordenados alfabéticamente, y se espera a que el usuario escriba el siguiente caracter.

- Caso de uso: Ver listas del usuario.
  - Descripción:El usuario verá las listas que tengan creadas, con las películas que haya añadido a cada una de ellas.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado.
  - Flujo de acontecimientos:
    1. El usuario accede a esta vista mediante el item de menú de la izquierda de la pantalla.
    2. Se cargan las listas de películas existentes asociadas al usuario.
  - Postcondiciones: Ninguna

- Caso de uso: Crear nueva lista.
  - Descripción: El usuario crea una lista a la que después podrá añadir películas.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado y situado en la vista que muestra las listas del usuario.
  - Flujo de acontecimientos:
    1. El usuario pulsa el botón de crear nueva lista.
    2. Se completa el campo con el nombre de la lista.
    3. Se pulsa el botón de crear lista.
  - Postcondiciones: Si se ha completado de manera correcta el campo del nombre de la lista, se crea y se muestra en pantalla.

- Caso de uso: Editar título de lista existente.
  - Descripción: El usuario edita el nombre de una lista existente.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado y situado en la vista que muestra las listas del usuario.
  - Flujo de acontecimientos:
    1. El usuario pulsa el botón de editar una lista.
    2. Se completa el campo con el nombre de la lista.
    3. Se pulsa el botón de confirmación.
  - Postcondiciones: Si se ha completado de manera correcta el campo del nombre de la lista, se guarda y se muestra en pantalla.

- Caso de uso: Eliminar lista.
  - Descripción: El usuario elimina una lista existente.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado y situado en la vista que muestra las listas del usuario.

  - Flujo de acontecimientos:
    1. El usuario pulsa el botón de eliminar lista.
    2. Se abre el cuadro de confirmación.
    3. Si se confirma se elimina la lista, si se cancela, se regresa a la pantalla en la que estaba el usuario.
  - Postcondiciones: Ninguna.

- Caso de uso: Ver listas de la comunidad.
  - Descripción: El usuario verá las listas que hayan creado los demás usuarios de la aplicación.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado.
  - Flujo de acontecimientos:
    1. El usuario accede a esta vista mediante un item de menú de la izquierda de la pantalla.
    2. Se cargan las listas de películas de los usuarios.
  - Postcondiciones: Solo se verán las listas de personas que no sean el usuario que ha iniciado sesión

- Caso de uso: Conocer una película sugerida por la aplicación.
  - Descripción: El usuario tendrá la opción de pedir al sistema que le sugiera una película, que podría ser interesante para el usuario
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado.
  - Flujo de acontecimientos:
    1. El usuario accede a esta vista mediante un item de menú de la izquierda de la pantalla, con el nombre "Sorpréndeme".
    2. Se verá un botón de acción en la pantalla.
    3. Al pulsar el botón el sistema mostrará una película al azar, cuya valoración sea mayor a 5 y haya tenido al menos 1000 votos.
    4. Una vez cargada la película, se podrá acceder a la información de la película, para poder añadirla a una lista o valorarla.
  - Postcondiciones: Si se vuelve a pulsar el botón de "Sorpréndeme", se muestra una película diferente.

- Caso de uso: Ver detalles de película.
  - Descripción: El usuario verá los detalles de la película seleccionada.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado.
  - Flujo de acontecimientos:
    1. El usuario accede a esta vista mediante el botón que aparece en cada película dentro de las listas del usuario, de la comunidad y las listas de películas de la pantalla de inicio, o mediante el buscador de películas o la pamtalla "Sorpréndeme".
    2. Se carga la vista con la película seleccionada y la información de la misma

  - Postcondiciones: Si la película ha sido valorada anteriormente por el usuario, se muestra dicha valoración.

- Caso de uso: Valorar película.
  - Descripción: El usuario puede valorar cualquier película que haya seleccionado.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado y situado en la vista que muestra los detalles de una película seleccionada.
  - Flujo de acontecimientos:
    1. El usuario pulsa el botón para valorar la película.
    2. El usuario puede valorar una película, mediante un selector del uno al diez.
    3. Se selecciona la valoración y se hace click en valorar, para enviar la valoración.
  - Postcondiciones: Se oculta el selector y se vuelve a mostrar el botón para volver a valorar la película si se desea.

- Caso de uso: Añadir película a lista.
  - Descripción: El usuario puede añadir cualquier película a una lista.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado y situado en la vista que muestra los detalles de una película seleccionada y debe tener mínimo una lista creada.
  - Flujo de acontecimientos:
    1. El usuario pulsa el botón de añadir a una lista.
    2. Aparece la lista desplegable con las listas del usuario.
    3. Se selecciona la lista en la que se quiere añadir la película.
    4. Se pulsa el botón añadir a lista.
  - Postcondiciones: La película se guarda en la lista seleccionada.

- Caso de uso: Eliminar película de una lista.
  - Descripción: El usuario elimina una película de una lista.
  - Actor: Usuario con sesión iniciada.
  - Precondiciones: El usuario debe estar autenticado y situado en la vista que muestra las listas que tenga asociadas. Además debe haber al menos una película en la lista.
  - Flujo de acontecimientos:
    1. El usuario sitúa el puntero del ratón sobre la película a eliminar.
    2. Aparece el botón de eliminación de la película.
    3. Se pulsa el botón y se abre una ventana de confirmación.
    4. Si se confirma la acción la película desaparece de la lista. si se cancela, se regresa a la vista anterior.
  - Postcondiciones: La película desaparece de la lista en cuestión.

CAPÍTULO 5
Análisis Conceptual y Diseño

Teniendo en cuenta el diseño de la aplicación se debe idear una manera de gestionar los datos de los usuarios, las listas de peliculas que se crean en base de datos, las valoraciones que se crean de cada una de las películas y las relaciones entre todos estos datos. En este capítulo se explica desde el diseño de la base de datos, la implementación del mismo en *SQL Server* y el diseño de la interfaz de la aplicación.

5.1 Diseño conceptual de la base de datos

Lo primero que se ha realizado es el diseño conceptual de la base de datos:

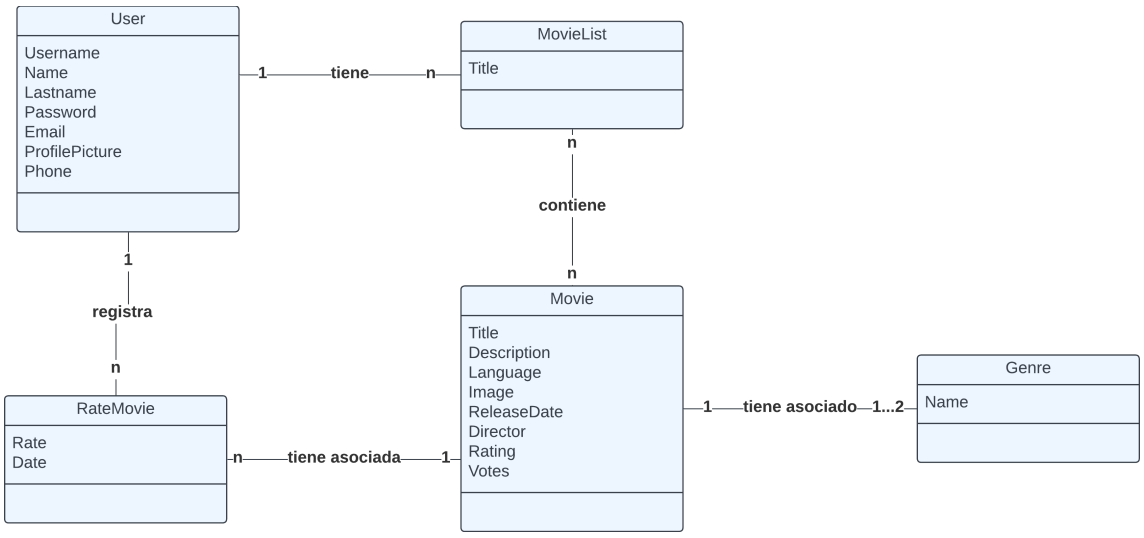

Figura 5.1: Diseño conceptual de la base de datos

Este diagrama, expresa, si lo leemos de izquierda a derecha, como un usuario, puede crear muchas listas , las cuales podrán tener asociadas muchas películas, y a su vez estas películas pueden estar en varias listas a la vez. También se puede ver rápidamente que un usuario puede crear una valoración para cada película y una película puede tener muchas valoraciones asociadas. Por último, también se puede ver que las películas tienen asociados entre uno y dos géneros cinematográficos.

5.2 Modelo de base de datos

Para entrar en detalle, a continuación se describen con mayor profundidad las diferentes tablas que formarán parte de la base de datos:

- *Users*: Se guardan los usuarios y sus datos para que puedan acceder a la aplicación utilizando un nombre de usuario y una contraseña
  - *Id*: *Id* del usuario en la base de datos
  - *Username*: Nombre o "nickname"del usuario identificativo dentro de la aplicación
  - *Password*: Contraseña del usuario.
  - *Name*: Nombre real del usuario.
  - *Lastname*: Apellidos del usuario.
  - *Email*: Correo electrónico del usuario.
  - *ProfilePhoto*: Foto de perfil elegida por el usuario.
  - *Phone*: Teléfono del usuario.
- *Movies*: En esta tabla se guardarán las películas que se hayan buscado en la *API*, para poder guardar sus datos de manera local, de manera que popdamos asociar un Id dentro de la base de datos de la aplicación y poder relacionarla con las listas y los usuarios.
  - *Id*: *Id* de la película en la aplicación
  - *Title*: Nombre de la película
  - *Description*: Descripción o pequeño resumen de la película
  - *Language*: Idioma en el que está producida la película
  - *Image*: Poster o carátula de la película
  - *ReleaseDate*: Fecha de estreno de la película
  - *Director*: Director de la película.
  - *Rating*: Valoración media de la película.
  - *Votes*: Número de votos de la película
  - *MovieApiId*: *Id* de la película en la *API* TMDB
  - *Genre1Id*: *Id* del género principal de la película.
  - *Genre2Id Id* del género sevundario de la película.
- Genres: Esta tabla guarda los géneros de las películas.
  - *Id*: *Id* del género en esta base de datos
  - *Name*: Nombre del género
  - *ApiId*: *Id* del género en la *API* TMDB.
- *MovieLists*: Esta tabla guarda para cada registro una lista, la cual tiene un nombre y va asociada a un usuario en concreto.
  - *Id*: *Id* de la lista de películas
  - *Title*: Nombre de la lista de películas
  - *UserId*: *Id* del usuario que creó la lista.

- *UserListMovie*: Esta tabla sirve para representar la relación de n:n entre dos tablas, en este caso las tablas *Users* y *MovieLists*. Con esto podemos representar en la base de datos la capacidad de un usuario de crear una lista que contenga muchas películas y que una película pueda estar en diferentes listas.
  - *Id*: *Id* de la relación de una lista con una película.
  - *MovieListId*: *Id* de la lista de películas.
  - *MovieId*: *Id* de la película asociada a la lista.
- *RateMovie*: Esta tabla almacena las valoraciones de los usuarios para cada película.
  - *Id*: *Id* de la valoración.
  - *Rate*: Valoración de la película.
  - *Date*: Fecha de la valoración.
  - *UserId*: *Id* del usuario que realiza la valoración.
  - *MovieId*: *Id* de la película valorada.

Teniendo definidas las tablas, ahora se debe tener la capacidad de gestionarlas en una base de datos, para esto se utilizan las siguientes herramientas:

- *SQL Server*: Esta tecnología se utiliza para gestionar la base de datos de la aplicación, puesto que ofrece diferentes ventajas a la hora de crear una base de datos desde cero, como por ejemplo, mecanismos de autenticación y autorización para acceder a ella, pero sobretodo por la capacidad de integrarse con el entorno de desarrollo de *.NET*[1], que en este caso es la tecnología base de la parte *back end* del proyecto. Para tener una mejor interacción con la tecnología de *SQL Server* se ha utilizado la herramienta *Microsoft SQL Server Management Studio 19*, dado que es uno de los sistemas más usados para esta tecnología, y existía experiencia previa en el uso de la misma.
- *Entity Framework*: Se trata de un ORM, que posibilita mapear objetos de código, en este caso desarrollado en *.NET*, a una base de datos relacional [10]. Esta abstracción de la base de datos a un modelo conceptual más cercano al lenguaje *C#*, hace el desarrollo más fácil en algunos aspectos como por ejemplo a la hora de dejar de lidiar con consultas SQL para el acceso y gestión de los datos almacenados en *SQL Server*. Para utilizar esta tecnología simplemente hay que instalarla en el proyecto *.NET* de la aplicación, desde el repositorio de Microsoft.
- *LINQ*: Es una tecnología que permite a los usuarios utilizar sentencias de código de lenguajes de programación de alto nivel, en este caso *C#*, en lugar de utilizar sentencias SQL, para gestionar la información almacenada en una base de datos relacional. Esta tecnología fue creada para *.NET*, razón por la que es muy útil en este proyecto. En el capítulo de Desarrollo de la solución 6.3.2, se comprobará que mediante las sentencias *LINQ* obtienen, crean y actualizan la información almacenada en la base de datos de la aplicación.

En el capítulo de Desarrollo de la solución 6.3.2 se explica con mayor detalle la implementación de la base de datos y la manera en la que se usan estas herramientas para llevarla a cabo.

Después de la correcta implementación mencionada, el modelo de base de datos visto desde la herramienta Microsoft *SQL Server Management Studio 19* será el siguiente:

[1] *.NET*: plataforma de código abierto para crear aplicaciones multiplataforma que se pueden ejecutar en cualquier sistema operativo.

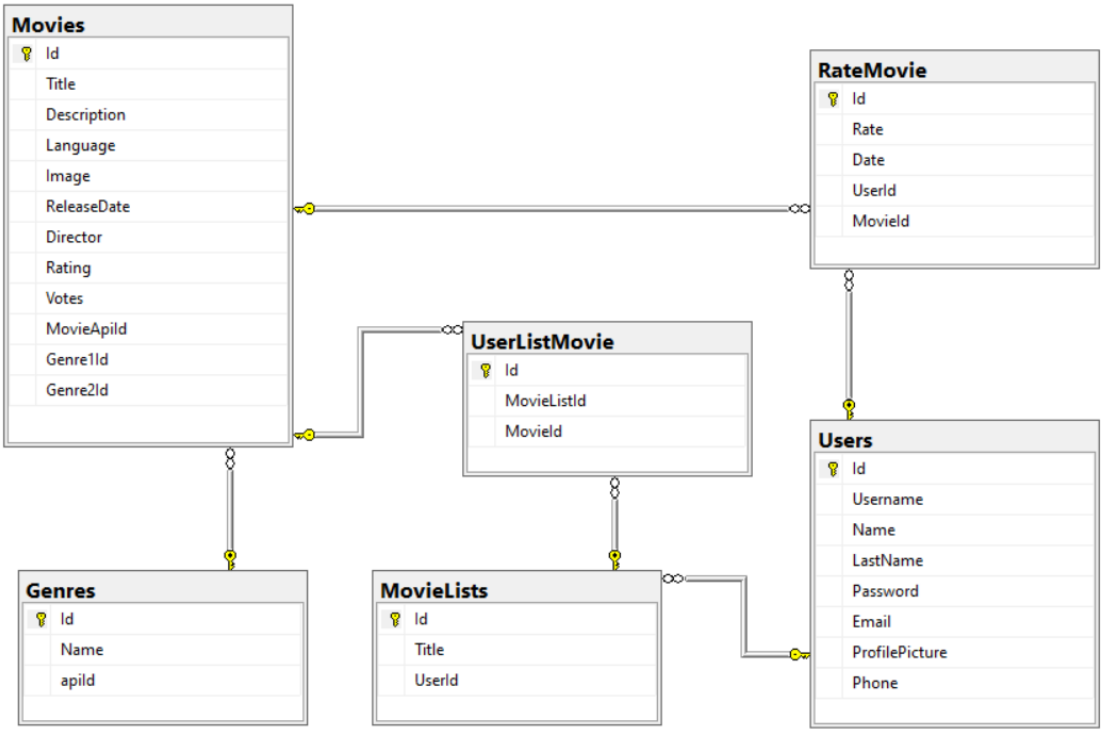

Figura 5.2: Diagrama de clases de la base de datos *SQL Server*

Y se puede ver como ejemplo, la tabla *Movies*, como ejemplo para ver su contenido, una vez transcurrido parte del desarrollo:

5.3 Diseño de la interfaz

Con los casos de uso definidos en el anterior capítulo y la base de datos ya creada, se puede comenzar con el diseño de la interfaz de usuario que se va a implementar. Esta es una parte importante, para tener en cuenta cuando se comience el desarrollo del *front end* en *Vue.js*. Para llevar a cabo el diseño de las vistas o pantallas, se ha considerado la idea de que los usuarios en todo momento comprendan la pantalla en la que están, las acciones que pueden realizar, los datos que están leyendo y en caso de que el usuario falle o cometa algún error, pueda en todo momento cancelar la acción.

Los diseños de interfaces son los siguientes:

- Iniciar sesión: En esta pantalla, el usuario ve un formulario, que le permite acceder a la aplicación en caso de estar registrado, o registrarse en caso de no estarlo.

- Registrarse: En esta pantalla una persona puede crear una nueva cuenta de usuario, y por tanto registrarse en la aplicación. También puede cancelar la acción y no completar el registro.

- Pantalla principal: En esta vista, el usuario ya autenticado, puede ver el menú principal, la barra de búsqueda que permanecerá visible a lo largo de toda la aplicación y desde el cual puede acceder a las demás pantallas. Aparte de estos componentes principales, se muestran las listas de películas más populares, en estreno, próximos estrenos y mejor valoradas.

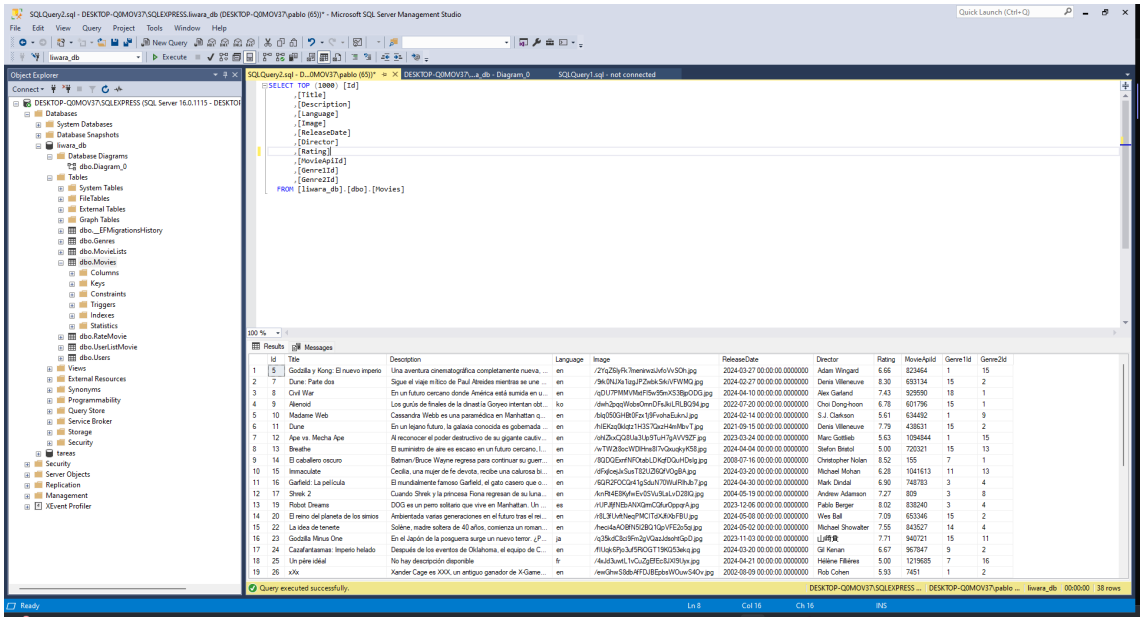

Figura 5.3: Tabla Movies desde la herramienta de gestión de BD

Figura 5.4: Diseño de la pantalla de inicio de sesión

Figura 5.5: Diseño de la pantalla de registro de un nuevo usuario

Figura 5.6: Diseño de la pantalla principal de la aplicación

- Pantalla "Mis Listas": Se muestran las listas que el usuario haya creado, y las opciones de crear una nueva lista o editar las existentes, ya sea eliminando películas de la lista, o cambiando el título de la lista.

Figura 5.7: Diseño de la pantalla que muestra las listas creadas por el usuario

- Pantalla "Listas de la comunidad": En esta pantalla el usuario podrá ver las listas que hayan sido creadas por otros usuarios, para poder ver las películas que han sido de interés para la comunidad.

Figura 5.8: Diseño de la pantalla que muestra las listas creadas por la comunidad

- Pantalla "Sorpréndeme": En esta vista, se muestra un botón, que al pulsarlo, permite a los usuarios recibir una película como sugerencia, ya sea para que un usuario la añada a una de sus listas o la pueda ver y luego valorarla.

Figura 5.9: Diseño de la pantalla *Sorpréndeme*, para descubrir nuevas películas

- Pantalla para ver la información de una película: En esta vista se visualiza la información más relevante de una película seleccionada, ya sea desde el buscador habilitado en todas las pantallas, o desde cualquiera de las listas de la aplicación.

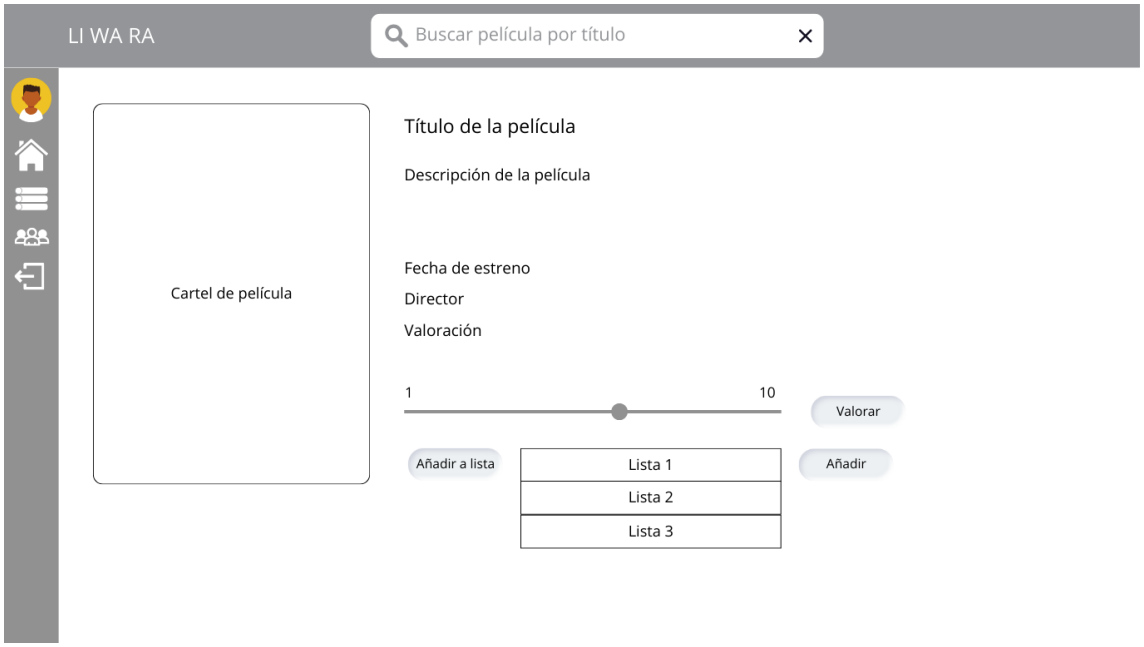

Figura 5.10: Diseño de la pantalla que muestra la información de una película

CAPÍTULO 6
Desarrollo de la solución

Para comenzar con el desarrollo del código de la aplicación, primero se debe elegir una arquitectura en la que basar el proyecto, de manera que se sigan unas pautas para organizar los elementos que formarán parte de la solución. A continuación se explica la decisión tomada, acerca de la arquitectura que se va a seguir, tanto en la parte *back end* de la aplicación, como en la parte *front end*. Además, se explica también la elección de las herramientas utilizadas.

6.1 Arquitectura de la aplicación

En este proyecto se utiliza la arquitectura basada en tres niveles. Dicha arquitectura distingue, como su nombre indica tres diferentes niveles o capas, que están separadas también en diferentes niveles lógicos y físicos:

- Nivel de datos: El nivel que se encarga de almacenar los datos de los usuarios, de las listas de películas, de las propias películas y de sus relaciones, y de suplir a la capa de aplicación de dichos datos cuando son solicitados. Haciendo que no se guarden datos en una capa que no corresponde. Para esta capa es muy importante el uso de *SQL Server*, tecnología anteriormente mencionada, para la implementación de la base de datos de la aplicación.
- Nivel de presentación (*front end*): Esta es la capa que se encarga de mostrar los datos al cliente, en este caso los datos guardados en la aplicación y los que provienen de la *API* TMDB, que provee la información de las películas. Esta capa es la que muestra una interfaz visual y accesible, para que el usuario pueda interactuar con dichos datos. En este nivel la principal tecnología utilizada será *Vue.js*, como ya se había comentado en capítulos anteriores.
- Nivel de aplicación (*back end*): En este nivel, es en el que se reciben las peticiones y la información desde el nivel de presentación, para poder procesarlos y contrastarlos con los datos provenientes de la capa de datos, y mediante el uso de la lógica de negocio correspondiente, devolver la información correcta. Además, se encarga de realizar actualizaciones y cambios en el nivel de datos. En este nivel la principal tecnología utilizada es *.NET* 8.0, en esto se hace mayor hincapié en el apartado de desarrollo de *back end* 6.3.

Además de esta estructura de tres niveles cabe mencionar que la aplicación hace uso de los datos de la *API* de información cinematográfica TMDB, en este caso se trata de un servicio *API*, del que se utilizan los datos de películas, imágenes y personas relacionadas con dichas películas. Esta *API*, hace muy fácil encontrar la información que necesita

Figura 6.1: Arquitectura en 3 capas o niveles

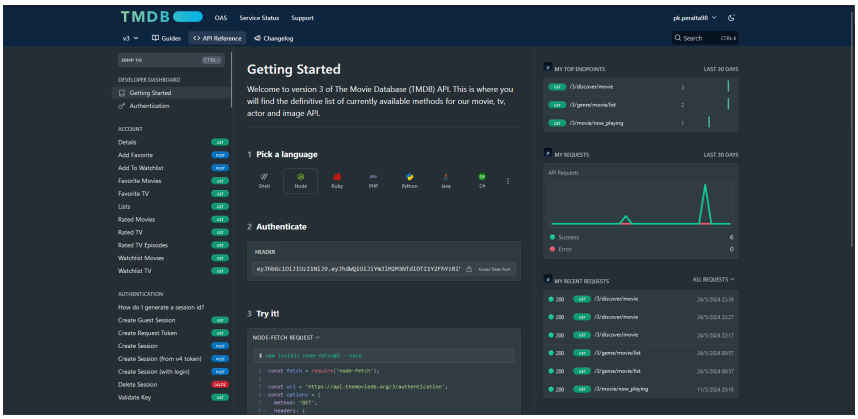

Figura 6.2: Vista de guía de la uso de *API* de TMDB

la aplicación. El uso de los datos de TMDB, es totalmente gratuito, en caso de ser un proyecto como este de ámbito académico. Es por eso la elección de esta *API*, ya que con simplemente registrarse en la página web, se puede obtener un *token*[1] que permita el acceso desde la aplicación. Otra de las razones de utilizar los datos de TMDB, es la guía de uso que posee y la documentación, que es muy completa y clara para cualquier usuario interesado en utilizarla.

6.2 Herramientas utilizadas

Ahora, que ya está definida la arquitectura a seguir en el desarrollo, se deben elegir las herramientas que ocasionarán un buen desempeño en la programación del código del proyecto.

Las herramientas son las siguientes:

- *Visual Studio 2022*: Se trata del *IDE*[2] por excelencia cuando se habla del desarrollo en el entorno de *.NET*. Cuenta con infinidad de herramientas que optimizan la labor de los desarrolladores, como *IntelliCode*, que proporciona sugerencias de código para hacer que la productividad aumente. También permite depurar de manera muy efectiva la ejecución de las aplicaciones, lo cual permite tener un control sobre los errores y excepciones del código, que pocas herramientas ofrecen. Incluso ofrece un gran soporte para diferentes proyectos.

[1] *token*: Cadena de caracteres aleatorios, que tienen significado en un entorno adecuado.
[2] *IDE*: También llamado Entorno de desarrllo integrado, es un sistema *software* para desarrollo de aplicaciones

- *Visual Studio Code*: Esta herramienta es una de las más populares, al ser considerado un *IDE* muy ligero, rápido, multiplataforma y por contar con una gran cantidad de extensiones y herramientas que facilitan y optimizan mucho los desarrollos. En el caso de *Vue.js*, *Visual Studio Code* también cuenta con diversas herramientas que hacen menos difícil de configurar un proyecto creado con esta tecnología, como por ejemplo la posibilidad de usar *npm*[3] desde la consola del propio VS Code, o el uso de *ESlint*, una herramienta para poder depurar nuestro código de manera fácil y sencilla.
- Postman: Esta herramienta es clave a la hora de probar el funcionamiento correcto de la parte *front end* de la aplicación, puesto que permite conectarse a las *url* asociadas a los métodos de los controladores y hacer peticiones, inserciones, actualizaciones y eliminaciones, sin necesidad de haber desarrollado la parte *front end* de la aplicación. Es muy útil para comprobar que los datos que se envían y se reciben son correctos.

Figura 6.3: Logos *Visual Studio 2022*, *Visual Studio Code* y *Postman*

Además de estas herramientas, para mantener el código a salvo, y no solamente en el ámbito local, se ha utilizado *GitHub*, que permite alojar el código desarrollado en la nube y control de versiones, para una buena gestión de cambios en la aplicación, este herramienta es muy útil sobretodo en desarrollos más grandes y en los que interviene más de un desarrollador. Los repositorios son:

Back end: `https://github.com/pabloperalta98/front-end-liwara.git`

Front end `https://github.com/pabloperalta98/back-end-liwara.git`

6.3 Desarrollo *back end*

Para esta aplicación es necesario el desarrollo de una parte *back end*, la cual se encarga de transformar la información almacenada en la base de datos en objetos que se pueden transportar a la parte *front end* de la aplicación. Para el desarrollo de esta parte del proyecto, se utilizan diferentes herramientas, diversas tecnologías y una lógica de negocio adecuada.

6.3.1. Tecnologías utilizadas

Las tecnologías utilizadas para desarrollar esta parte de la aplicación, son las siguientes:

- *.NET* 8.0: Es una plataforma que nos ofrece diferentes aplicaciones para poder desarrollar y ejecutar nuestros propios servicios web y nuestras propias aplicaciones.

[3]npm: gestor de dependencias de *Node.js*

Es de código abierto, multiplataforma y gratuito. Esta plataforma tiene como lenguaje predeterminado *C#*, el cual es un lenguaje fuertemente tipado y orientado a objetos. Además en esta versión esta plataforma ofrece un rendimiento mejorado, con tiempos de carga menores y además tiene una comunidad muy activa que nos permite resolver las dudas que tengamos en cualquier momento simplemente buscando información por foros y webs relacionadas de internet.

- JWT [11](*JSON Web Token*): Estándar abierto (RFC 7519) que define un formato compacto y autónomo para asegurar que la transferencia de datos entre dos partes sea segura. Se utiliza generalmente para autenticación [12] de usuarios en aplicaciones web, como en este caso. En esta aplicación, se utiliza de manera que cuando el usuario inicia sesión introduciendo su nombre de usuario y su contraseña, en ningún momento de la transacción entre *back end* y *front end* de estos datos, se transporta la contraseña sin codificar. Mediante esta práctica, se mantiene protegida la información del usuario mientras está autenticado en la aplicación.

  Además de estas tecnologías, se ha utilizado también, para la interacción con la base de datos, *Entity Framework*, como se ha indicado en el capítulo de Análisis conceptual y diseño 5.2.

6.3.2. Lógica de negocio aplicada

La lógica de negocio se encarga de transformar las reglas de negocio del mundo real en código que nos permita determinar cómo la información se puede crear, almacenar o cambiar. Para el *back end* de esta aplicación, la lógica de negocio se ha estructurado de la siguiente manera:

- Entidades: En este apartado se crean las clases que corresponden íntegramente a las tablas de la base de datos, de manera que sean mucho más amigables para el desarrollo, las transacciones realizadas entre base de datos y *back end*. Para una mejor comprensión de la estructura de dichas entidades, se muestra la clase *MovieList* como ejemplo:

```
1  public class MovieList
2  {
3      public int Id { get; set; }
4      [Required]
5      [StringLength(50)]
6      public string Title { get; set; }
7      public int UserId { get; set; }
8      [ForeignKey("UserId")]
9      public virtual User User { get; set; }
10 }
```

  Teniendo en cuenta el ejemplo, se puede ver que los atributos de esta entidad, tienen asociados valores que corresponden a clases del espacio de nombres de *.NET DataAnnotations*, los cuales nos permiten definir los metadatos de dichos atributos. Para esta aplicación se han aplicado las siguientes características a los atributos de las clases:

  - Required: Esta característica, hace que el atributo, obligatoriamente tenga que ser diferente de nulo, dando lugar a que el usuario no pueda crear un objeto con este atributo vacío.
  - StringLength: Como indica el nombre de la característica, permite que sea posible controlar la longitud de la cadena introducida en el atributo de la clase.

- ForeignKey: Esta característica, permite definir cuando uno de los atributos es una clave ajena de otra entidad.

En la entidad, el atributo Id, se lee directamente como un campo de identidad y que por tanto será autonumérico. Esto es debido al nombre del atributo. Si el nombre fuera otro, habría que indicar mediante la propiedad *[Key]* de *DataAnnotations* que se trata de una clave primaria o identificador. Una vez creadas las entidades con sus respectivas características y atributos, se crea la clase *ApplicationDBContext*, la cual hereda de la clase *DBContext* [13], que se utiliza para configurar la conexión a la base de datos, definir las entidades que se almacenarán en ella y aplicar cambios. Además esta clase hace uso también de la clase *DBSet*, para definir las tablas de la base de datos mediante la creación de propiedades que utilizan las entidades definidas.

```
public class ApplicationDBContext : DbContext
{
    public ApplicationDBContext(DbContextOptions<ApplicationDBContext>
        options) : base(options)
    {
    }
    public DbSet<User> Users { get; set; }
    public DbSet<UserListMovie> UserListMovie { get; set; }
    public DbSet<MovieList> MovieLists { get; set; }
    public DbSet<Genre> Genres { get; set; }
    public DbSet<Movie> Movies { get; set; }
    public DbSet<RateMovie> RateMovie { get; set; }
}
```

Para aplicar esta estructura a la base de datos real, se utilizan comandos en la consola *NuGet* [14], que permiten migrarla al entorno de *SQL Server*.

Una vez se crea la migración, también se genera una clase provista por la herramienta *Entity Framework*, con las propiedades de la base de datos que se han definido para dicha migración, y es aquí donde si hay algún error, se puede modificar la configuración de las entidades.

```
public partial class v10 : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {

        migrationBuilder.CreateTable(
            name: "Users",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Username = table.Column<string>(type: "nvarchar(50)",
                    maxLength: 50, nullable: false),
                //Se verían el resto de atributos de cada clase de la BD
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Users", x => x.Id);
            });
    // La clase continua con todas las entidades que se han declarado en
        AplicationDBContext
    }
}
```

Tras comprobar que la configuración a migrar es la correcta, se aplica a la base de datos real.

- Servicios: En los servicios es donde se encuentra toda la lógica que se debe aplicar a los datos que provienen de la base de datos. Por tanto, en este punto es en el que se recibe la información del servidor de base de datos, se aplica la lógica definida y se preparan los datos que se van a enviar al *front end*. A continuación se muestra el ejemplo de uno de los servicios de la aplicación, en concreto el servicio *MovieListService*, que permite obtener las listas de películas creadas por el usuario:

```
namespace back_end_liwara.Services
{
    public interface IMovieListService
    {
        IEnumerable<MovieList> GetMovieLists(int userId);
        MovieList GetMovieList(int id);
        MovieListUserModel CreateMovieList(MovieList movieList);
        MovieList UpdateMovieList(int id, string title);
        MovieList DeleteMovieList(int id);
        //Se declara el resto de métodos necesarios del servicio
    }
    public class MovieListService : IMovieListService
    {
        //Declaración del objeto _context, que nos da el contexto de
            datos, con la información obtenida desde base de datos SQL
            Server
        private readonly ApplicationDBContext _context;

        public MovieListService(ApplicationDBContext context)
        {
            _context = context;
        }

        //Método para obtener las listas de un usuario(sin las películas
            que contienen)
        public IEnumerable<MovieList> GetMovieLists(int userId)
        {
            return _context.MovieLists.Where(x => x.UserId == userId).
                ToList();
        }
        //La clase continúa con la implementación del resto de métodos
            declarados en la interfaz
}
```

Todos los servicios tienen esta estructura, con:

- Interfaz: Como se puede ver en el ejemplo, la práctica de tener una interfaz, hace posible que haya una separación entre lo que hace un servicio y el cómo lo hace, ya que en la interfaz se declaran los métodos que se van a utilizar y por tanto las acciones que va a realizar el servicio. Esto ocasiona una mayor claridad de código, haciendo que se más maleable y comprensible.
- Clase: En la clase, se implementa la interfaz declarada al principio del servicio, es decir, se definen las acciones que va a realizar cada método y las funcionalidades que ello implica. Además de implementar los métodos de la interfaz, se pueden declarar nuevos métodos que no sean accesibles desde la interfaz y que sirvan para factorizar código o para ser usados únicamente en el ámbito de dicha clase. En el ejemplo de código, se ve implementado 1 de los 11 métodos declarados en la interfaz. Algunos de los métodos son para hacer una lectura de los datos desde el servidor de base de datos, como *GetMovieLists*. Para obtener los datos que cumplan las condiciones requeridas, se hace uso de consultas *LINQ*, característica explicada en el apartado de Análisis concep-

tual y diseño 5.2, mediante esta característica se realizan las consultas con la sintaxis siguiente:

```
return _context.MovieLists.Where(x => x.UserId == userId).ToList();
```

En esta consulta, se utiliza el contexto de datos leído desde base de datos, y mediante el método *Where* de *LINQ*, se filtran los resultados, con la condición de que los objetos tengan para el campo *UserId* de la tabla *MovieLists*, el valor del parámetro *userId* y por último, se hace uso del método *ToList*, para que el resultado de la consulta sea mapeado a un objeto *List*, el cual es más fácil de utilizar en un entorno *.NET*. En dichos casos no se realiza una actualización de la base de datos, ya que no se cambia el contenido de ninguna tabla, sin embargo, en el caso de los métodos de actualización, creación y eliminación se hace uso de métodos *LINQ* como *Add* para añadir elementos, *Entry* para actualizar elementos, *Delete* para eliminar elementos y *SaveChanges* para guardar los cambios realizados en el contexto actual, en la base de datos real. Finalmente si los cambios son salvados de manera correcta, se devuelven los datos que resultan de aplicar la lógica del método, al controlador que esté usando el servicio.

- Controladores: En este punto es en el que se reciben las llamadas desde *front end*, solicitando datos o enviando información a contrastar con la lógica definida en los servicios. Los controladores se encargan de recibir esas peticiones y de enviar el resultado de dicha petición al *front end*. A continuación se muestra el ejemplo del controlador que se encarga de recibir y gestionar las peticiones relativas a listas de películas:

```
namespace back_end_liwara.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class MovieListsController : ControllerBase
    {
        //Se declara el servicio MovieListService, ya que se va a hacer uso de sus métodos y la consecuente lógica implementada
        public IMovieListService _movieListService;
        public MovieListsController(IMovieListService movieListService)
        {
            _movieListService = movieListService;
        }
        //Método que gestiona la solicitud de las listas de un usuario
        [HttpGet]
        [Route("GetMovieLists")]
        public async Task<IActionResult> GetMovieLists(int userId)
        {
            try
            {
                var response = _movieListService.GetMovieLists(userId);
                return Ok(response);
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }

        }
        //El controlador completo implementa todos los métodos que recibirán llamadas desde el fornt end
    }
}
```

En el controlador se define primero la ruta de acceso al controlador, mediante la propiedad siguiente:

```
[Route("api/[controller]")]
```

Esto hace que la ruta sea:

https://rutadelservidordebackend/api/nombredelcontrolador.

Para acceder a los métodos que gestionan las peticiones *HTTP*, el proceso es parecido, ya que se accede mediante dos propiedades, la primera se encarga de reconocer el tipo de petición, el cual en el caso de esta aplicación puede ser:

- *Get*: En caso de ser una petición de lectura de datos.
- *Post*: En caso de ser una petición de creación de datos.
- *Put*: En caso de ser una petición de actualización de datos.
- *Delete*: En caso de ser una petición de eliminación de datos.

La segunda propiedad será la ruta que da acceso a dicho método. Podemos ver las dos propiedades en el siguiente ejemplo, correspondiente al método *GetMovieLists*:

```
[HttpGet]
[Route("GetMovieLists")]
public async Task<IActionResult> GetMovieLists(int userId)
```

Aparte de la ruta de acceso correcta y la petición adecuada, se tiene que enviar desde el servidor que hace la petición, los objetos y valores de los parámetros definidos en los métodos del controlador. Teniendo en cuenta los valores de dichas propiedades y los datos que deben ser enviados junto a la petición, la ruta de acceso a este método sería para el usuario cuyo Id sea 1:

https://rutadelservidordebackend/api/MovieLists/GetMovieLists?userId=1.

Todos los métodos del controlador utilizan los servicios y sus funcionalidades, para aplicar la lógica correspondiente a las peticiones y devolver los datos correctos al *front end*.

6.4 Desarrollo del *front end*

El *front end* de una aplicación es la parte encargada de mostrar los datos de manera visual, interactiva y accesible a los usuarios. Esta parte de la aplicación se desarrolla generalmente utilizando tecnologías como *HTML*, *CSS* y *JavaScript*, las cuales colaboran para dar a la aplicación la funcionalidad necesaria para que la experiencia de los usuarios, sea óptima.

6.4.1. Tecnologías utilizadas

Para realizar el desarrollo del *front end* de la aplicación se han utilizado las siguientes tecnologías:

- *Node.js*: Se trata de un entorno de ejecución de *JavaScript*. Este entorno, permite ejecutar código en el lado del servidor en lugar de en el navegador. Para desarrollar un proyecto en *Vue.js* se vuelve una herramienta muy útil ya que permite utilizarlo como servidor de desarrollo. También ofrece la funcionalidad de gestionar las

dependencias de tu proyecto. Mediante *npm*, se pueden instalar bibliotecas y herramientas clave para un proyecto, como en mi caso es *Vue Router* [15], *Pinia*, *Vuetify* [16] y sobretodo *Vue CLI*, que permite crear el proyecto de *Vue.js* desde cero. Este entorno de ejecución, también tiene la funcionalidad de poder compilar el código desarrollado para poder implementarlo en un entorno de producción, en este caso no se ha hecho uso de esta característica, ya que el proyecto está más centrado en el desarrollo que en la implementación de una aplicación en una fase de producción, más centrada en poner la aplicación de cara a los usuarios reales y clientes.

- *Vue.js* [17]: es un *framework*[4] de *JavaScript* de código abierto que permite a los desarrolladores crear interfaces de usuario de una sola página (*SPA*), tiene una curva de aprendizaje baja, si se tiene experiencia en desarrollos con *JavaScript*. Tiene la opción de ser utilizada mediante la inclusión de sus dependencias a través de *CDN*, pero en el caso de este proyecto, se ha hecho una instalación de las dependencias utilizando *npm* y se ha creado el proyecto *Vue.js* desde 0. Este *framework* proporciona el uso de archivos de extensión .vue, los cuales se dividen en tres partes y tienen siempre la misma estructura:

  - *Template*: En esta parte de la estructura se define la estructura *HTML* del componente y en consecuencia, es en la que se declaran los componentes que se van a ver en la pantalla, sean componentes *HTML*, *Vue.js*, *Vuetify* o componentes personalizados creados por la persona que está desarrollando la aplicación.
  - *Script*: Aquí se declara la lógica que va a seguir el componente, incluyendo sus propiedades, métodos, ciclos de vida y cualquier otra funcionalidad.
  - *Style*: En esta parte es en la que se declaran los estilos que va a adoptar el componente que se está desarrollando. Esta sección es opcional y puede contener *CSS*, *SCSS* o cualquier preprocesador de *CSS*.

  Además del *framework Vue.js* se ha utilizado diversas herramientas que tienen un desempeño clave en diferentes funcionalidades de la aplicación:

  - *Vuetify*: Biblioteca de componentes de interfaz de usuario para *Vue.js*, que proporciona un gran número de componentes que los desarrolladores pueden utilizar para crear aplicaciones web, que sean atractivas y funcionales. Estos componentes tienen una alta capacidad de personalización y además de todos ellos existe una documentación completa. Aparte de estas características, *Vuetify* cuenta con una comunidad activa que proporciona ayuda y soluciones a problemas que se puedan tener al utilizar esta tecnología. Esta opción es elegida por delante de otras como *Bootstrap*, ya que es específica de *Vue.js*.
  - *Vue Router*: Es la biblioteca principal de enrutamiento de *Vue.js* y proporciona a las aplicaciones que la usan, una buena gestión de la navegación por diferentes vistas y componentes, mediante las rutas de las mismas. Esas rutas se definen en un archivo .js que tiene una estructura como la que se ve a continuación:

```
1 import { createWebHistory, createRouter } from 'vue-router'
2 import { useAuthStore } from '@/stores';
3 import HomeView from '@/components/HomeView.vue'
4 import LogIn from '@/components/LogIn.vue'
5
6 //Ahora se crea el objeto routes, que contiene las rutas con el
    nombre de la vista dentro del router, la ruta que tendrá y el
    componente que se está cargando para esa ruta
7 const routes = [
8   { path: '/', name: 'home', component: HomeView },
```

[4] *Framework*: conjunto de reglas para el desarrollo de *software* de manera eficiente

```
  { path: '/login', name: 'login', component: LogIn}
]
//Ahora se crea el historial que se va a utilizar en la aplicación,
    para navegar por ella correctamente.
const router = createRouter({
  history: createWebHistory(),
  routes,
})
//Método de comprobación de usuario autenticado
router.beforeEach(async (to) => {
  //Se redirige a login si el usuario no está autenticado
  const publicPages = ['/login'];
  const authRequired = !publicPages.includes(to.path);
  const auth = useAuthStore();
  if (authRequired && !auth.user) {
      auth.returnUrl = to.fullPath;
      return '/login';
  }
});
export default router
```

- *Pinia*: Es una solución a la gestión del estado de aplicaciones *Vue.js*. Con esta herramienta se pueden definir *stores* que permiten guardar estados de la aplicación, es decir, valores cargados que se van leer en los componentes de la aplicación, en este caso se utiliza para mantener guardado el estado de un usuario, es decir, si ha iniciado sesión o no en la aplicación y poder acceder a los datos guardados en esa *store* o incluso modificarlos, desde cualquiera de las pantallas o vistas. Es una opción muy ligera y más intuitiva que la anteriormente más utilizada que es *Vuex*, la cual se encarga también de la gestión de estados pero tiene mayor dificultad de aprendizaje y un único *store* centralizado para todos los componentes, en lugar de poder crear tantos *stores* como se crea conveniente que es el caso de esta herramienta.

- *Axios* [18]: Es una biblioteca de *JavaScript* utilizada para realizar las solicitudes *HTTP*, en este caso desde *Node.js*, tanto a la parte *back end* de la aplicación, como a la *API* de TMDB de la que se obtienen los datos de las películas y las listas de películas que se muestran en la pantalla de inicio. Algunas de sus principales características son:

  - Puede hacer *XMLHttpRequests* desde el navegador
  - Puede hacer peticiones *HTTP* desde *Node.js*
  - Intercepta petición y respuesta
  - Transforma petición y datos de respuesta
  - Cancela peticiones
  - Transformacion automatica de datos *JSON*[5]
  - Soporte para proteger al cliente contra *XSRF*

6.4.2. Componentes y estructura

Se han utilizado componentes de *Vue.js* y *Vuetify* para crear las diferentes pantallas o vistas de la aplicación, en algunos casos se han creado componentes personalizados para poder reutilizarlos en diferentes pantallas y con datos diferentes, consiguiendo así un mejor uso del código dessarrollado. Además para gestionar los estados de inicio de

[5] *JSON*: Formato de texto, de intercambio de objetos en *JavaScript*

Figura 6.4: Logos *Node.js*, *Vue.js*, *Vuetify*, *Pinia* y *Axios*

sesión se han utilizado *stores* de acciones. Por lo tanto se ha dividido el código en cuatro claras secciones:

- Componentes: Los componentes son piezas reutilizables de la interfaz de usuario, que permiten dividir la aplicación en partes más pequeñas y manejables, promoviendo así la modularidad y la reutilización de código. Como se ha comentado anteriormente, en esta aplicación, los componentes utilizados son provistos por el *framework Vuetify*. Para ver todo esto de manera más clara, a continuación se muestra código que corresponde al componente *DataIteratorSearchMyLists*, desarrollado para mostrar una lista de películas del usuario. En este caso se muestra la parte *Template* del componente, dado que es la que se encarga de la interfaz que será mostrada al usuario.

```
<template>
  <v-card>
    <!--Este componente se encarga de recibir la lista de películas, la
        cual se encuentra dentro de la propiedad itemsList->
    <v-data-iterator
      :items="itemsList"
      :items-per-page="6"
      :search="search"
      :title="title"
    >
      <!--Si fuera necesario, se mostraría un buscador de películas
          dentro de dicha lista->
      <template v-if="enableSearch" v-slot:header>
          <!--Aquí van los componentes que forman dicho buscador-->
      </template>

      <template v-slot:default="{ items }">
        <!--Aquí se mostrarán las películas->
      </template>
      <!--Se muestra un selector de páginas para ver todas las películas
          de la lista->
      <template v-slot:footer="{ page, pageCount, prevPage, nextPage }">
        <!--Aquí van los botones para cambiar de página-->
      </template>
    </v-data-iterator>
  </v-card>
</template>
```

- Vistas: Cuando se hace referencia a las vistas, se está hablando de las pantallas de la interfaz, con las que el usuario intaractúa. Estas vistas se han creado a partir de componentes de *Vuetify* y componentes personalizados, dado que dicha manera de trabajar hace que el desarrollo sea mucho más sostenible, reutilizable y fácil de cambiar.

Ahora, para visualizar mejor lo que es una vista, se muestra el contenido de *MisListasView.vue*, que se encarga de mostrar las listas creadas por un usuario, y de las acciones que se pueden hacer sobre cada una, además de crear alguna nueva.

```
<template>
  <v-card-title class="text-h4">Mis listas</v-card-title>
  <v-card>
        <!--Se muestra el botón de agregar una nueva lista y los demás
            botones y componentes que aparecerán al realizar esta acción-->
    <v-card-actions>
      <v-btn v-if="!showAddListField" @click="showAddList()">A adir
          nueva lista</v-btn>
      <v-text-field  v-if="showAddListField" v-model="newListName" label=
          "Nombre de la lista"></v-text-field>
      <v-btn v-if="showAddListField" @click="addNewList()">Confirmar</v-
          btn>
      <v-btn v-if="showAddListField" @click="cancelAddList()">Cancelar</v
          -btn>
    </v-card-actions>
        <!--Se declara el componente DataIteratorSearchMyLists y se le
            pasan los datos que va a mostrar aen la interfaz->
    <data-iterator-search
      <!--Se asignan los valores necesarios para cada propiedad que tiene
          el componente-->
    />
  </v-card>
</template>
```

Viendo el código anterior, es posible darse cuenta de que la creación del componente *DataIteratorSearchMyLists*, nos da una serie de ventajas importantes:

- Limpieza de código: Esto es posible gracias a que todo el código que corresponde al componente que se declara, se encuentra en el archivo .vue de dicho componente, de otra manera estaría declarado en el mismo archivo *MisListasView.vue*.
- Mejor organización: Otra de las ventajas de tener esta estructura de trabajo, es que el desarrollo se organiza mejor, de manera que hay una carpeta que contiene solamente las vistas y otra que contiene solamente los componentes.

- *Stores*: En este apartado se contienen los datos del usuario que deben ser accesibles cada vez que se necesiten en cualquier punto de la aplicación. Esto hace que sea más fácil la utilización de datos persistentes, ya sea por ejemplo el nombre de usuario, el id del usuario dentro de la base de datos, o el *token JWT* del usuario.

```
import { defineStore } from 'pinia';
import router from '@/router';
import axios from 'axios';

const baseUrl = 'https://localhost:44370/api/Users';

export const useAuthStore = defineStore({
    id: 'auth',
    state: () => ({
        // Se inicializa el estado desde el almacenamiento local para
            permitir al usuario permanecer conectado
        user: JSON.parse(localStorage.getItem('user')),
        returnUrl: null,
        users: {}
    }),
    actions: {
        async login(username, password) {
```

```
            const user = await axios.post(`${baseUrl}/authenticate`, {
                username, password });
            // Se actualiza el estdo de Pinia
            this.user = user;
            //Se almacenan los datos de usuario y el token jwt en el
                almacenamiento local el navegador, para mantener al
                usuario conectado entre las actualizaciones de la página
            localStorage.setItem('user', JSON.stringify(user));
            //Se redirige a la url anterior o por defecto a la página de
                inicio
            router.push(this.returnUrl || '/');
        }
        //A continuación se declararían más acciones como registrarse y
            cerrar sesión.
    }
});
```

- Constantes: Se trata de un archivo *JavaScript*, que contiene información que será constante para todo la aplicación web. Es por eso que en este archivo se definen las rutas de acceso a la *API* de TMDB, al *back end* y de *front end*, además de el *token* de acceso a la *API* de TMDB.

```
export const TMDB_AUTH = {
    key: 'bbbe3c757b925caac4bc85ff93527336',
    url: 'https://api.themoviedb.org/3/'
};
export const BACK_API = {
    url: 'https://localhost:44370/api/'
};
export const FRONT_URL = {
    url: 'http://localhost:8080/'
};
```

6.4.3. Interacción con *API* TMDB

Para la interacción de la aplicación web con la *API* de datos de TMBD, se utilizan peticiones *HTTP* a través de la herramienta *Axios*, las cuales hacen una consulta y mediante el *token* de acceso obtenido al registrar un usuario, se obtienen los datos en formato *JSON*.

Para ejemplificar las llamadas a la *API*, se muestra a continuación la manera en la que se obtienen las listas de películas que se ven en la página de inicio de la aplicación, se hace la petición a la *API* y se muestran los datos que vienen directamente desde ella.

```
<script>
  import axios from 'axios';
  import {TMDB_AUTH} from '@/constants/constants.js';
  import DataIteratorSearch from './DataIteratorSearch.vue';
  export default {
    components: {
      DataIteratorSearch,
    },
    data: () => ({
      search: '',
      itemsListPopulares: []
    }),
    created() {
      this.popularMovies();
    },
    methods: {
      //Este metodo recoge la lista de películas populares
      popularMovies() {
```

```
19          const url = TMDB_AUTH.url + 'movie/popular?api_key=' + TMDB_AUTH.key +
                '&language=es-ES';
20          axios.get(url)
21            .then(response => {
22              var popularMoviesList = response.data.results;
23              this.itemsListPopulares = popularMoviesList;
24              console.log(response.data);
25            })
26            .catch(error => {
27              console.error(error);
28            });
29        },
30        //A continuación se declararían los métodos de obtención de las otras
              listas de la pantalla principal.
31  </script>
```

Como se puede ver hay un método para obtener cada lista. En él hay una llamada *GET* de *HTTP* mediante *Axios* que obtiene la lista de películas desde la *API* de TMDB. Se le indica tanto la *url*, como el *token* que se necesita para acceder a los datos. Una vez realizada la petición, se obtiene la respuesta, que contiene la información solicitada en formato *JSON*.

En los componenetes de *Vue.js*, existen ciclos de vida [19], es decir los componentes se crean, se cargan, se insertan en la vista y después se destuyen. En cada uno de estos ciclos, se puede ejecutar código. Depende del caso, conviene más la ejecución de código en uno u otro ciclo.

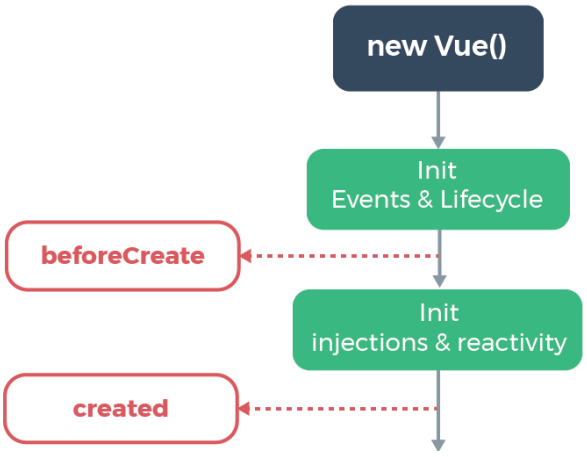

Figura 6.5: Primeros ciclos de un componente *Vue.js*

Para poder utilizar los datos que traen los métodos de una vista o componente, se hace uso del ciclo de vida *created()*, de manera que se obtenga la información, antes de que sea cargada y renderizada la vista, consiguiendo que no haya ningún error, que impida en ciclos posteriores que la vista muestre los datos correctos.

6.4.4. Interacción con el *back end*

En cuanto a la interacción del *front end* con el *back end*, las transacciones entre estas dos partes se llevan a cabo mediante llamadas a los controladores que nos proporcionan los datos solicitados en formato de objetos *JSON*. A través de esta interacción, se obtienen los datos del usuario, las listas de películas, los datos de las películas asociadas a dichas listas y las valoraciones de los usuarios de las películas.

Para ver de manera clara dicha interacción, a continuación se verá el código de los métodos que se utilizan para hacer las peticiones en la pantalla que muestra las listas de películas de un usuario.

```
created() {
    this.movieListsUrl = BACK_API.url + 'MovieLists';
    this.user = useAuthStore().$state.user.data;
    this.GetFilledListsFromUser();
    this.lists = useMovieListStore().$state.movieLists;
},
methods: {
    //Se recogen las listas del usuario con sus películas.
    async GetFilledListsFromUser(){
        const movieListStore = useMovieListStore();
        var self = this;
        await axios.get(`${this.movieListsUrl}/GetFilledMovieLists?userId=`+
            self.user.id)
        .then(response => {
            this.lists = response.data;
            console.log(this.lists);
            movieListStore.$state.movieLists = this.lists;
        })
        .catch(error => {
            console.error(error);
        });
    }
    //A continuación se declararían el resto de métodos que interaccionan con
        el back end.
}
```

Para realizar cada petición se ha accedido al *back end* mediante los controladores y las rutas que se han definido en dichos controladores y sus métodos.

También en este caso, como en el de interacción con la *API* TMDB, se hacen llamadas a los métodos en el ciclo de vida *created()*. Una vez se ha realizado la petición, se espera la respuesta que permite que los datos obtenidos de la lectura, creación, actualización o eliminación, se vean reflejados en la vista.

CAPÍTULO 7
Producto desarrollado

Tras el desarrollo de la aplicación, ahora se muestran diferentes capturas de la proyecto terminado. Para cada captura se explica a continuación como interactuar con la interfaz y como los usuarios reciben las respuestas de dichas interacciones por parte de la aplicación.

- Pantalla de inicio de sesión: Será lo primero que se vea en caso de que el usuario no haya iniciado sesión, haya cerrado sesión o no tenga cuenta y se quiera registrar. Se ven dos campos de texto, en los que se debe introducir el nombre de usuario y la contraseña que se haya dado de alta anteriormente. Una vez rellenos los campos si estos han sido completados de manera correcta, se iniciará sesión, de lo contrario, no se podrá acceder a la aplicación. Si el usuario no estuviera registrado, podría apretar el botón Registrarsez crear un usuario nuevo.

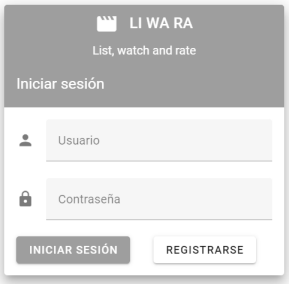

Figura 7.1: Pantalla de inicio de sesión

- Pantalla de registro: En esta pantalla se crea un nuevo usuario, introduciendo todos los datos que se piden en los campos de texto y eligiendo uno de los dos avatares de los que se dispone. Una vez completados todos los campos obligatorios, se procede a la creación del usuario, pulsando el botón 'Registrarse'. En caso de haber accedido a esta pantalla por error, o simplemente en caso de querer cancelar la creación y registro de un nuevo usuario, se presiona el botón de 'Cancelar'.

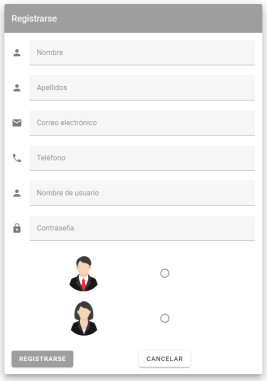

Figura 7.2: Pantalla de registro de nuevo usuario

- Pantalla principal: Una vez el usuario ha iniciado sesión, accede a esta pantalla. En esta pantalla se muestran cuatro listas de películas: las más populares actualmente, las películas en cartelera, próximos estrenos y las mejor valoradas. Además de estas listas es posible ver en la izquierda un menú que contiene los accesos a las demás vistas y funciones de la aplicación.

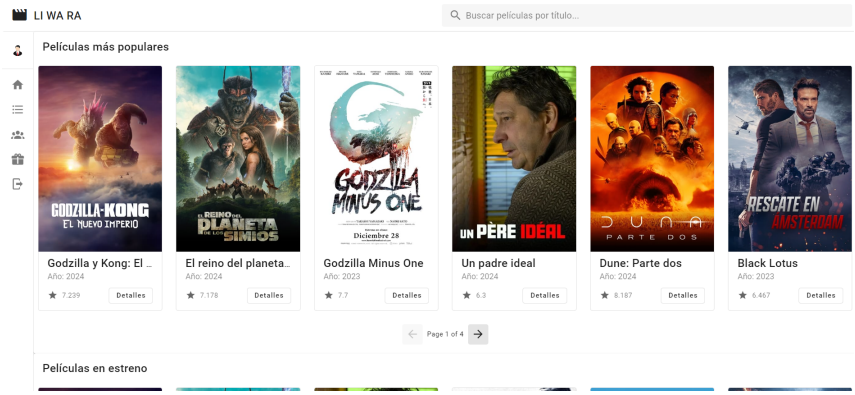

Figura 7.3: Pantalla principal

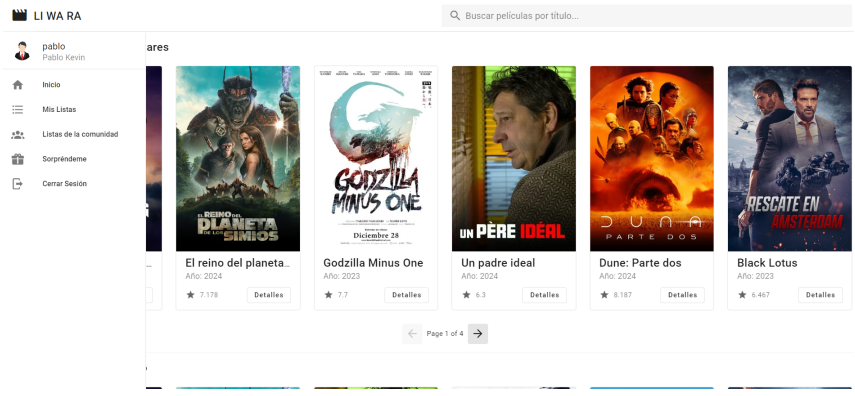

Figura 7.4: Menú de acceso a otras vistas

- Pantalla de "Mis Listas": En esta pantalla se muestran las listas que tiene el usuario creadas. Si tienen películas asignadas aparecerán en la lista correspondiente. También será posible añadir una nueva lista. Para esto se usará el botón de 'Añadir nueva lista', que nos mostrará un campo de texto en el que se introduce el nombre

de la lista a crear y se puede guardar esa nueva lista o cancelar la acción si se desea. Además de crear una nueva lista también será posible eliminar cualquier lista que se haya creado, así como editar el nombre de una de las listas o eliminar una película de una de las listas.

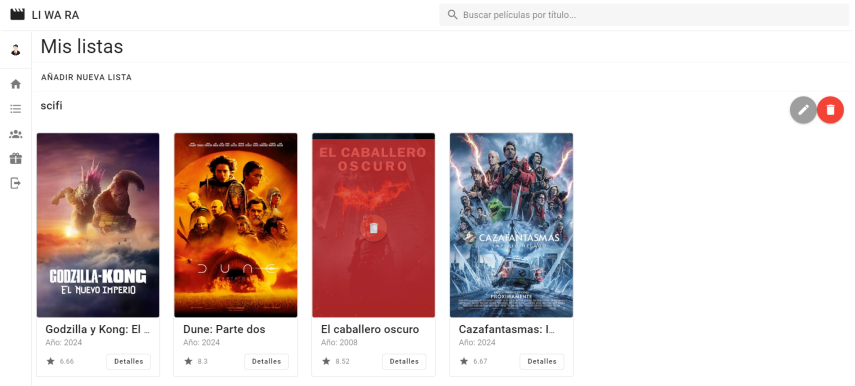

Figura 7.5: Pantalla de Mis Listas

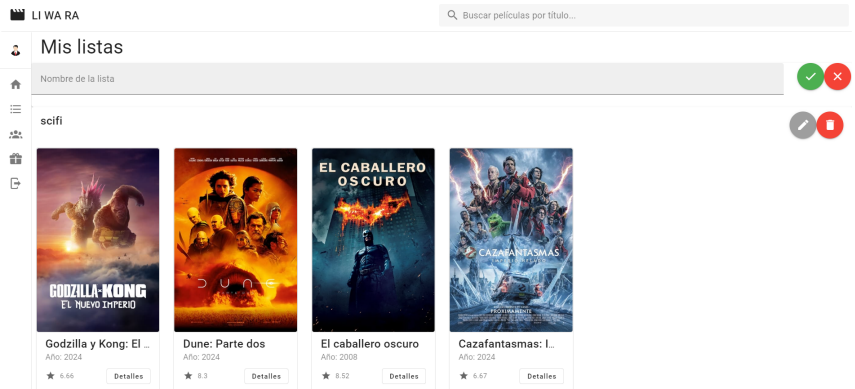

Figura 7.6: Pantalla Mis Listas: Añadiendo nueva lista

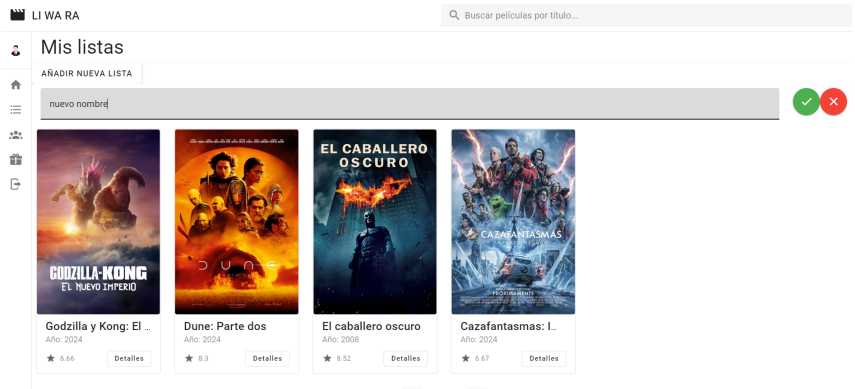

Figura 7.7: Pantalla Mis Listas: Editando lista

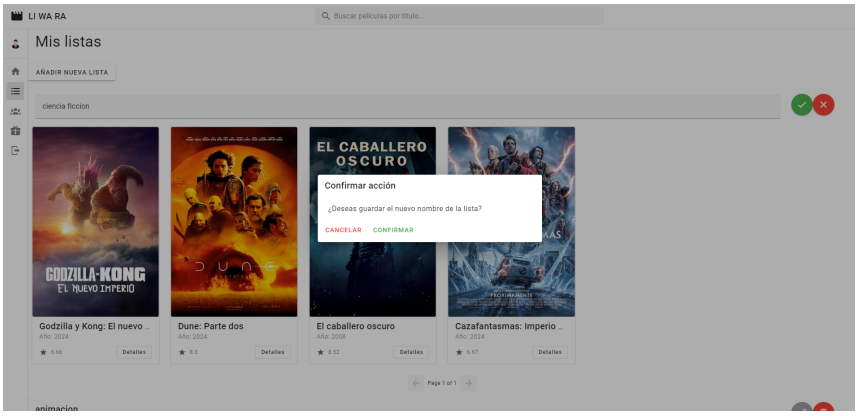

Figura 7.8: Pantalla Mis Listas: Confirmar edición de lista

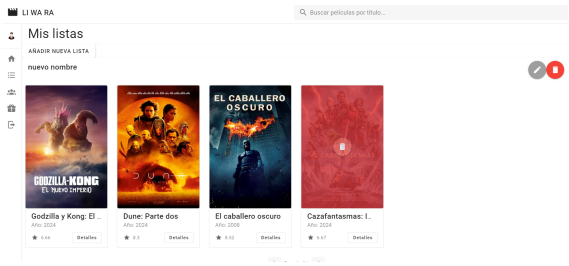

Figura 7.9: Pantalla Mis Listas: Eliminando película de una lista

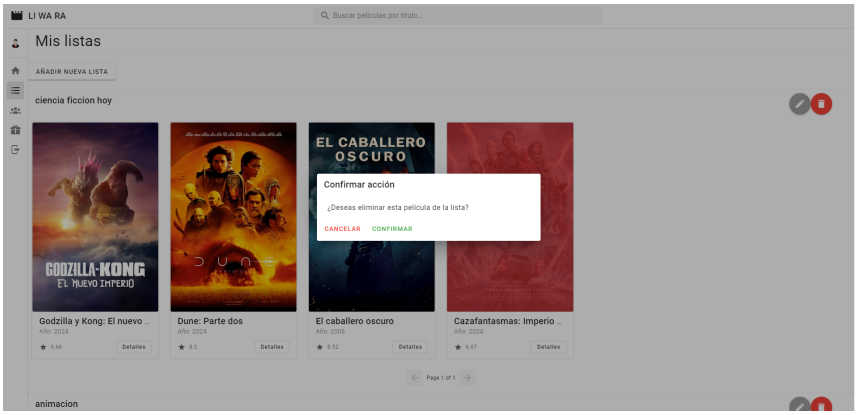

Figura 7.10: Pantalla Mis Listas: Confirmar eliminar película de una lista

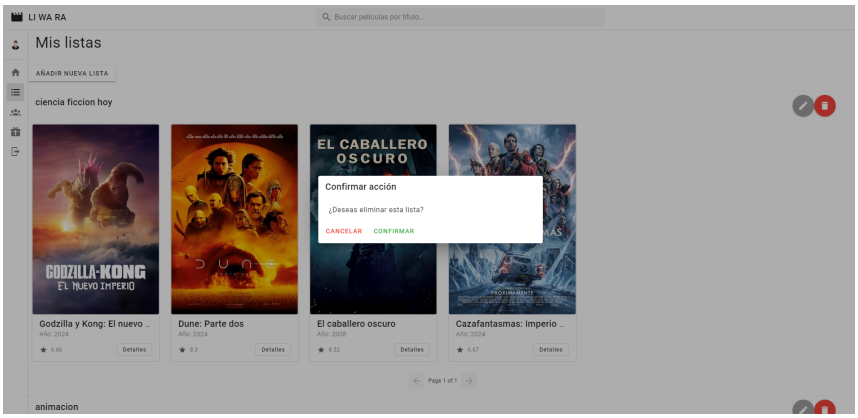

Figura 7.11: Pantalla Mis Listas: Confirmar eliminar lista

- Pantalla con información de una película: Esta pantalla es accesible cuando se quieren ver más detalles de una película seleccionada desde cualquiera de las listas, desde el buscador de películas o desde la pantalla "Sorpréndeme". Se pueden visualizar los datos de dicha película, la valoración que tiene y realizar algunas acciones sobre la película, como por ejemplo añadirla a una de las listas creadas, o poner una valoración a dicha película.

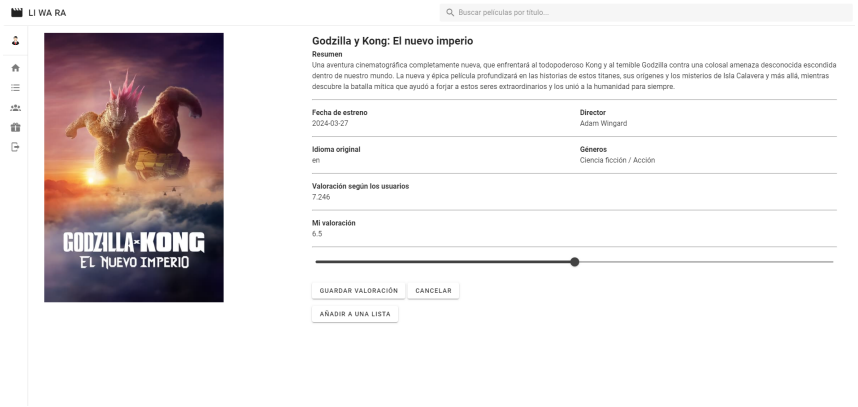

Figura 7.12: Pantalla de información de una película

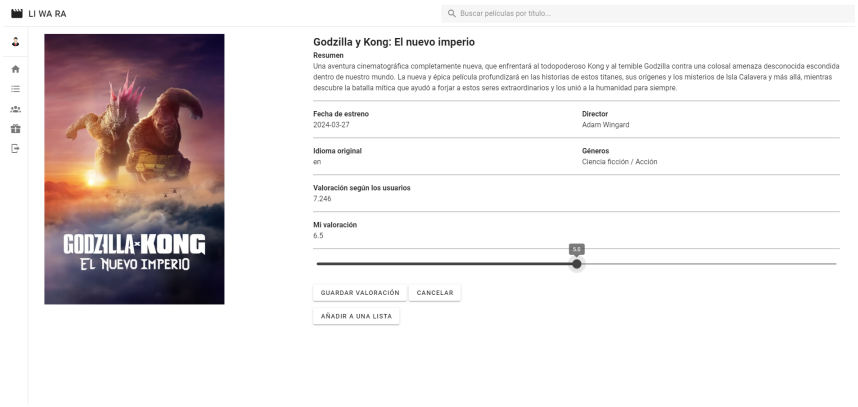

Figura 7.13: Pantalla de información de una película: Valorando película

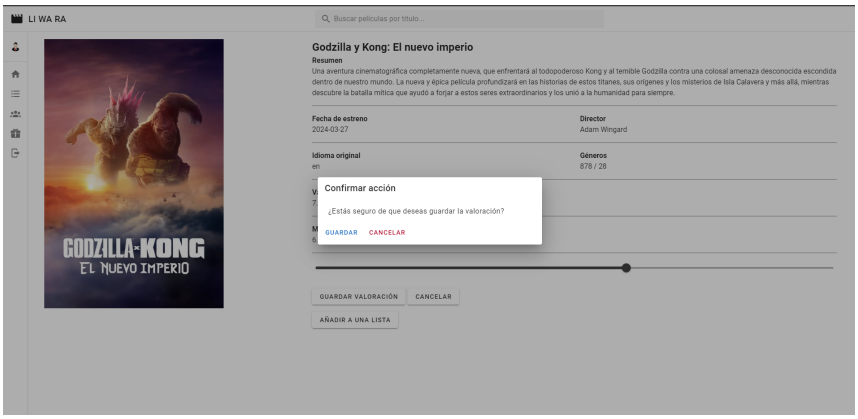

Figura 7.14: Pantalla de información de una película: Confirmar valoración de película

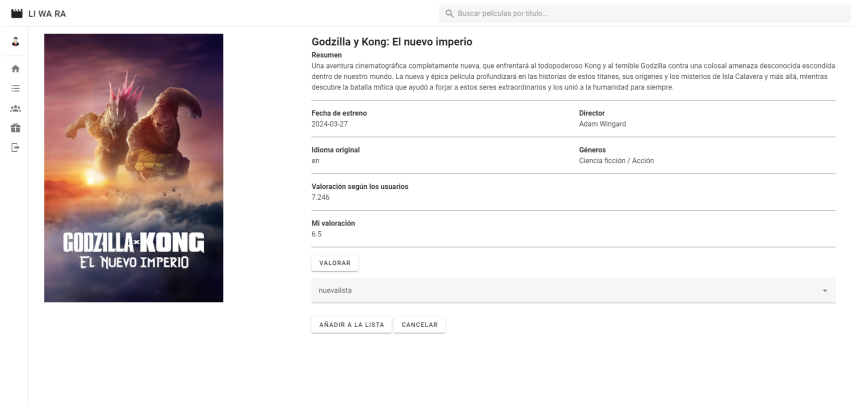

Figura 7.15: Pantalla de información de una película: Añadiendo película a una lista seleccionada

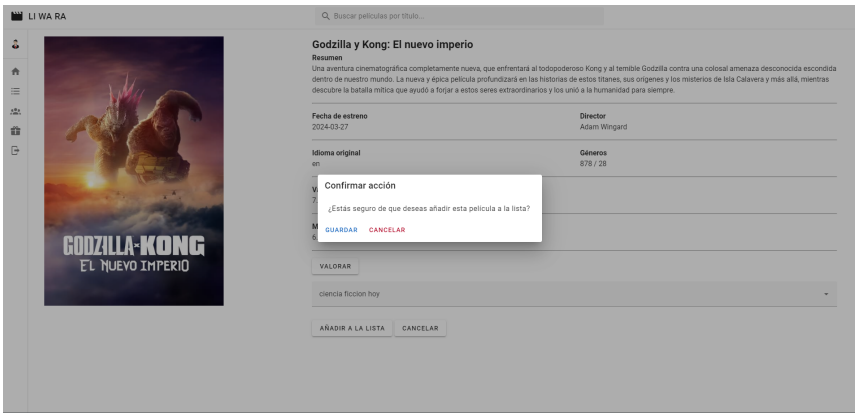

Figura 7.16: Pantalla de información de una película: Confirmar añadir película a lista

- Listas de la comunidad: En esta pantalla se ven las listas creadas por los demás usuarios de la aplicación.

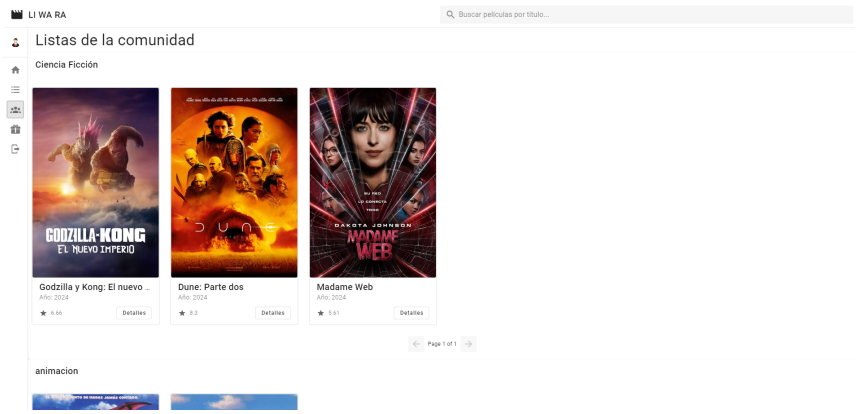

Figura 7.17: Pantalla en la que se ven las listas creadas por los usuarios

- Sorpréndeme: Esta pantalla muestra un botón que te sugiere una película que puedas ver, de manera aleatoria. Una vez pulsado el botón, se muestra una película al azar, cuya valoración sea de entre 5 y 10, y tenga al menos 1000 votos. Si el usuario desea otra sugerencia, puede volver a pulsar el botón "Sorpréndeme".

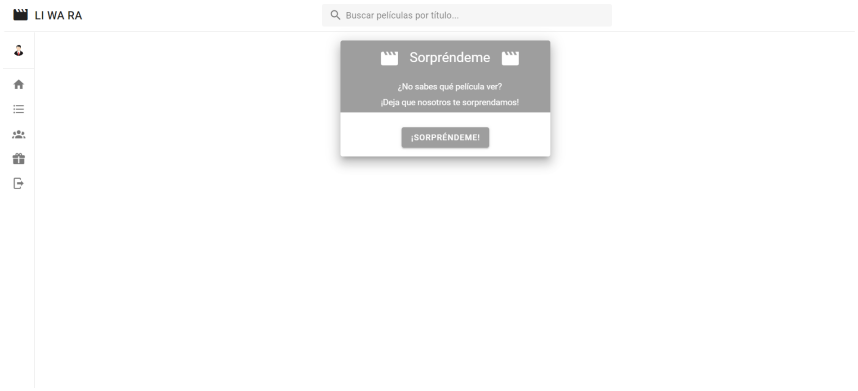

Figura 7.18: Pantalla "Sorpréndeme"

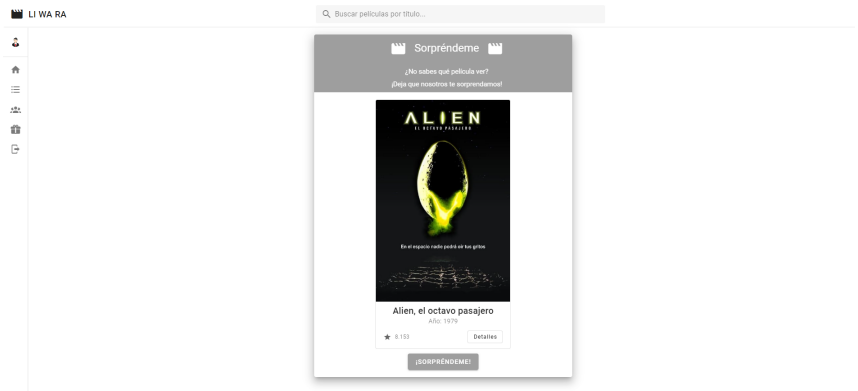

Figura 7.19: Pantalla "Sorpréndeme": Película sugerida tras pulsar el botón

CAPÍTULO 8
Validación

Para completar el desarrollo de una aplicación web, como la que se lleva a cabo en este proyecto, se debe realizar una validación, que compruebe que la interfaz cumple con ciertos estándares de uso de interfaces. Para ello se hace uso del "Decálogo de Principios Heurísticos de Jakob Nielsen-[20], el cual se puede definir en los siguientes puntos:

- Conocimiento del estado del sistema: El usuario debe ser consciente en todo moomento, en que pantalla se encuentra de la aplicación. En cada pantalla se indica no solamente mediante el título, la pantalla en la que se encuentra el usuario, sino mediante el menú, que indica la vista a la que se ha accedido.
- Relación clara entre el diseño de interfaz y el mundo real: La información debe mostrarse siguiendo las convenciones del mundo offline, con un orden natural y lógico. Se muestra el cartel o póster y el título de las películas, como cuando una persona va al cine y quiere decidir la película que va a ver.
- Usuario con capacidad de elegir lo que va a hacer: El usuario es libre de cancelar las acciones que realiza si lo necesita.
- Coherencia y estándares: Se debe seguir un estándar que se ajuste a *software* ya desarrollado. La interfaz sigue una línea de desarrollo que el usuario puede reconocer, si ha interactuado con cualquier otro sistema parecido anteriormente.
- Prevención de errores: Los errores que el usuario pueda cometer se tienen en cuenta y se manejan de manera que no interrumpan la interacción del usuario con el sistema. Para cada acción que conlleve una modificación de datos, como por ejemplo una creación de una lista, o eliminación de una película de una lista, existen ventanas de confirmación, que permiten cancelarlas en caso de fallo del usuario.
- Reconocimiento antes que memoria: no se debe obligar al usuario a tener que recordar cómo se utiliza la aplicación. En esta aplicación los botones de acción y de cambio de vista, asi como los formularios, tienen descripciones claras que hacen más fácil la utilización del sistema.
- Uso eficiente del sistema: Tanto los usuarios expertos como los novatos deben ser capaces de usar lo más eficientemente posible la aplicación. En esta aplicación, ninguna de las acciones es muy complicada, por tanto el uso es en cualquier caso muy fluido.
- Diseño simple y estético: Solamente se debe incluir lo relevante. No se debe mostrar información que pueda ser irrelevante para el usuario. Precisamente es una de las claves de este proyecto y solamente se muestran datos importantes y necesarios en cada una de las vistas, ya sea de películas o listas de películas.

- Ayudar al usuario a reconocer los errores, diagnosticarlos y recuperarse de ellos: los mensajes de error deben ser claros, informar sobre el problema y sugerir de forma constructiva una solución. En este caso la aplicación no ofrece funcionalidades que puedan dar lugar a un error. del usuario.
- Ayuda y guía al usuario: Aunque lo ideal es que el sistema pueda utilizarse sin necesidad de ayuda, en algunos casos, especialmente si el sistema es demasiado complejo es conveniente proveer al usuario de una guía de uso de la aplicación. Puesto que este sistema no es demasiado complejo, no es necesaria ayuda para eluso de la aplicación.

CAPÍTULO 9
Conclusiones

Tras completar el proyecto desarrollado, es posible llegar a ciertas conclusiones acerca del proceso de la realización del mismo. Desde el principio, lo primero que se hizo es definir la motivación del proyecto, y los objetivos que este va a proponer. Se define que el principal móvil de este trabajo es mejorar la gestión de la información cinematográfica que una persona amante del cine, tiene sobre las películas que quiere ver, ha visto o simplemente le recomiendan ver. Para ello se haría uso de listas en las que agrupar las películas, tantas como el usuario quiera crear, con los títulos cinematográficos que quiera. Una vez definida la motivación, se procede al análisis de aplicaciones que ya adopten las funcionalidades que se implementan en la aplicación propuesta. De esta manera se consigue información muy importante para el desarrollo de la solución. Un análisis así es totalmente necesario, ya que permite ver ejemplos de funcionalidades que se pueden aplicar a la aplicación que se está creando o funciones que distan de hacer que la aplicación sea sencilla y manejable para el usuario. Posterior a esto, se elige la metodología que se utiliza para trabajar en el proyecto, haciendo uso de un análisis y un estudio de las diversas opciones que existen. Esta parte ha sido clave, puesto que llevó a la elección de una metodología incremental, la cual se adapta de manera casi perfecta al desarrollo de una aplicación como esta. Tras la elección de la metodología se realiza la recogida de requisitos y los consecuentes casos de uso que se van a programar, mediante esta práctica sabemos de manera real el alcance que queremos afrontar en el desarrollo. Con toda esta información recogida, se procede al desarrollo de la solución, incluyendo la creación de una base de datos relacional de *SQL Server*, el desarrollo de una parte *back end* y una parte *front end*. Se especificaron todas las herramientas utilizadas y como interactúan las diferentes partes de la solución. Siendo el *back end* el que actúa como *API* para el *front end* haciendo la labor de proveedor de los datos del usuario, las listas que tiene dicho usuario, las películas que están asociadas a dicha lista y las valoraciones que hace de las películas. También se explica como el *front end* usa los datos de la *API* de TMDB, para visualizarlos directamente en la interfaz. Por último se comprueba el funcionamiento del proyecto finalizado mediante capturas de pantalla que sirven para ejemplificar de manera visual la interacción de un usuario con la aplicación. Además se incluye la validación de la aplicación y su interfaz haciendo uso del "Decálogo de Principios Heurísticos de Jakob Nielsen", y comprobando así de manera heurística que la aplicación creada cumple los requisitos necesarios para ser considerada una interfaz correcta. Algunas de las posibles futuras mejoras, podrían ser añadir una funcionalidad que permita a los usuarios tener amigos y poder ver las listas que estos amigos crean, podría añadirse la opción de hacer comentarios acerca de cada película, añadir filtros de búsqueda de películas o un buscador de listas de otros usuarios. Otra mejora sería llevar la aplicación a un entorno de producción, para que no sea accedida solamente de manera local, pese a que este TFG se centra más en el desarrollo que en la producción de cara a clientes reales. Todas estas me-

joras se pueden aplicar, pero teniendo precaución de no desviarse del objetivo principal de la aplicación.

Bibliografía

[1] *El cine y las plataformas de streaming, ¿amigos o enemigos?* Disponible en: https://welabplus.com/2021/08/11/cine-y-streaming-amigos-o-enemigos/ (Consultado: el 27 de mayo de 2024).

[2] *¿Qué es Back End, Front End y Back Office y por qué es importante para tu web?*. Disponible en: https://nestrategia.com/desarrollo-web-back-end-front-end/ (Consultado: el 20 de mayo de 2024)

[3] *Frontend ¿qué es y para qué se utiliza en desarrollo web?* Disponible en: https://www.arsys.es/blog/frontend-que-es-y-para-que-se-utiliza-en-desarrollo-web (Consultado: el 20 de mayo de 2024).

[4] *Metodologías de desarrollo de software: ¿qué son?* Disponible en: https://www.santanderopenacademy.com/es/blog/metodologias-desarrollo-software.html (Consultado: el 24 de mayo de 2024).

[5] *¿Qué es la metodología lean?* Disponible en: https://www.atlassian.com/es/agile/project-management/lean-methodology (Consultado: el 24 de mayo de 2024).

[6] *¿Qué Es Extreme Programming (XP)? - Valores, Principios Y Prácticas*. Disponible en: https://www.nimblework.com/es/agile/programacion-extrema-xp/ (Consultado: el 24 de mayo de 2024).

[7] *Qué es el modelo incremental*. Disponible en: https://blog.comparasoftware.com/que-es-el-modelo-incremental/ (Consultado: el 24 de mayo de 2024).

[8] *The Movie Database (TMDB)*. https://developer.themoviedb.org/docs/getting-started (Consultado: el 20 de mayo de 2024).

[9] *Guía definitiva para crear casos de uso - A nivel de requerimientos*. Disponible en: https://www.arkanapp.com/post.php?id=29 (Consultado: el 24 de mayo de 2024).

[10] *¿Qué es una base de datos relacional?* Disponible en: https://www.oracle.com/es/database/what-is-a-relational-database/ (Consultado: el 24 de mayo de 2024).

[11] *Qué es Json Web Token y cómo funciona*. Disponible en: https://openwebinars.net/blog/que-es-json-web-token-y-como-funciona/ (Consultado: el 27 de mayo de 2024).

[12] *Vue 3 + Pinia - JWT Authentication Tutorial & Example*. Disponible en: https://jasonwatmore.com/post/2022/05/26/vue-3-pinia-jwt-authentication-tutorial-example (Consultado: el 24 de mayo de 2024).

[13] *Qué es DBContext y DBSet*. Disponible en: https://www.netmentor.es/entrada/dbcontext-dbset (Consultado: el 24 de mayo de 2024).

[14] *¿Qué es NuGet y qué hace?* Disponible en: https://learn.microsoft.com/es-es/nuget/what-is-nuget (Consultado: el 25 de mayo de 2024).

[15] *Getting Started | Vue Router*. Disponible en: https://router.vuejs.org/guide/ (Consultado: el 25 de mayo de 2024).

[16] *Vuetify — A Vue Component Framework*. Disponible en: https://vuetifyjs.com/ (Consultado: el 24 de mayo de 2024).

[17] *Qué es Vue JS y qué lo diferencia de otros frameworks*. Disponible en: https://openwebinars.net/blog/que-es-vue-js-y-que-lo-diferencia-de-otros-frameworks (Consultado: el 24 de mayo de 2024).

[18] *Axios*. Disponible en: https://axios-http.com/es/ (Consultado: el 24 de mayo de 2024).

[19] *Aprende a usar el ciclo de vida de Vue (created, mounted, destroyed...)*. Disponible en: https://codingpotions.com/vue-ciclo-vida/ (Consultado: el 25 de mayo de 2024).

[20] *Metodología UX: Evaluación heurística mide la usabilidad de una interfaz*. Disponible en: https://www.mtp.es/blog/experiencia-de-usuario-blog/evaluacion-heuristica-la-usabilidad-una-interfaz/ (Consultado: el 24 de mayo de 2024).

APÉNDICE A

Anexo: ODS - Objetivos de desarrollo sostenible

Objetivos de Desarrollo Sostenible		Alto	Medio	Bajo	No procede
ODS 1	Fin de la pobreza.				X
ODS 2	Hambre cero.				X
ODS 3	Salud y bienestar.				X
ODS 4	Educación de calidad.				X
ODS 5	Igualdad de género.				X
ODS 6	Agua limpia y saneamiento.				X
ODS 7	Energía asequible y no contaminante.				X
ODS 8	Trabajo decente y crecimiento económico.				X
ODS 9	Industria, innovación e infraestructuras.	X			
ODS 10	Reducción de las desigualdades.				X
ODS 11	Ciudades y comunidades sostenibles.				X
ODS 12	Producción y consumo responsables.				X
ODS 13	Acción por el clima.				X
ODS 14	Vida submarina.				X
ODS 15	Vida de ecosistemas terrestres.				X
ODS 16	Paz, justicia e instituciones sólidas.				X
ODS 17	Alianzas para lograr objetivos.				X

Tabla A.1: Objetivos de Desarrollo Sostenible relacionados al TFG

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

La relevancia de un TFG centrado en el desarrollo de aplicaciones de gestión cinematográfica para los Objetivos de Desarrollo Sostenible (ODS) es alta en el caso de el objetivo 9, que corresponde a 'Industria, innovación e infraestructuras'. Este objetivo tiene como finalidad construir infraestructuras resilientes, promover la industrialización inclusiva y sostenible y apoyar la innovación. Una aplicación web de gestión de contenido cinematográfico, como la propuesta, puede contribuir a este ODS de diversas maneras.

Un ejemplo sería, contribuir a la mejora de la infraestructura de la industria cinematográfica proporcionando una plataforma que facilite la distribución y el acceso a las producciones cinematográficas que gozan de un menor alcance a nivel de publicidad, para llegar a un gran número de espectadores, o que solamente son promocionadas en un número limitado de países.

El desarrollo de una aplicación como la de este TFG, contribuye a que, no solamente los títulos cinematográficos menos conocidos se vean beneficiados, sino que ayuda a que los usuarios expandan sus gustos o preferencias en este sector promoviendo una mayor diversidad en el tipo de público que consume cada género fílmico, independientemente de la popularidad actual del mismo. Esta creciente diversidad provoca un aumento de audiencia en cada género, lo que en consecuencia comporta una mejora de la calidad de todas las nuevas producciones.

Otra manera de contribuir a este objetivo, es expandiendo el alcance de las producciones que son realizadas desde un ámbito artístico y de expresión de un mensaje que puede ser importante para las sociedades a nivel global, en lugar de servir solamente para ampliar el foco de las superproducciones con un mensaje vacío, cuyo único objetivo es el beneficio económico, teniendo un impacto directo en la economía de este sector, haciendo un poco más igualado el reparto de beneficios económicos.

Una aplicación así, permite además un avance en el ámbito de la gestión de datos, ya que personas que tuviesen organizadas las películas que eran de su interés en archivos desorganizados, tales como blocs de notas, hojas de excel, aplicaciones instaladas que no utilizan, que ocupan memoria de los dispositivos que utilizan y por lo tanto contribuyen a un gasto de materiales físicos y de energía, ahora optimizan los recursos que utilizan dado que la aplicación se encarga de ello.

En conclusión, esta aplicación contribuye al crecimiento, de la ya en auge, industria cinematográfica en el ámbito económico y en popularidad, dado que no solamente contribuye expandiendo las producciones con menos alcance, sino a todos los niveles de producciones, desde las que tienen un presupuesto bajo a las que tienen un presupuesto muy alto. Además de acercar el conocimiento del sector a un gran número de personas.