

IWATCH

“Descubre, disfruta, repite”

Alumnos: Andreu Linares y Joel Cosp
Curso: Desarrollo de aplicaciones web, 2023 - 2024

PRESENTACIÓN DEL PROYECTO

Este documento recoge cada una de las fases necesarias que se han realizado durante los últimos tres meses para llevar a cabo la creación de una página web totalmente funcional y apta para la gestión de una plataforma dedicada al entretenimiento y reproducción de contenido audiovisual..

El proyecto se desglosa a lo largo de un índice, donde se explica cada punto de forma detallada, desde el diseño previo del prototipo inicial de la web hasta el último fragmento de código.

PALABRAS CLAVE

- Aplicación
- Web
- Cine
- Películas
- Series
- Playlist
- Audiovisual
- Plataforma
- Suscripción
- Plan

KEYWORDS

- Application
- Web
- Cinema
- Movies
- Series
- Playlist
- Audiovisual
- Platform
- Subscription
- Plan

ÍNDICE

1. INTRODUCCIÓN	5
2. CONTEXTO Y MOTIVACIÓN	5
2.1. Objetivos generales de producción	5
2.2. Objetivos de aprendizaje	5
3. DESCRIPCIÓN	6
3.1. Resultados esperados	6
3.2. Beneficios	6
4. ANÁLISIS	6
4.1. Requisitos funcionales	6
4.2. Requisitos no funcionales	7
4.3. Requisitos de la base de datos	7
4.3. 1. Diseño de la base de datos	8
4. 3. 1. 1. Modelo entidad relación	8
4. 3. 1. 2. Modelo relacional	8
5. DISEÑO	9
5.1. Definición de la empresa e identificación del target	9
5. 1. 1. Identificación de la imagen corporativa de la empresa	9
5. 2. Wireframe	10
5. 2. 1. Registro de sesión	10
5. 2. 2. Inicio de sesión	11
5. 2. 3. Sala principal	11
5. 2. 4. Información de la película	12
5. 2. 5. Reproductor	12
5. 2. 6. Salas de cine	13
5. 2. 7. Sala	13
5. 2. 8. Perfil	14
5. 2. 9. Plan de suscripciones	14
5. 2. 10. Métodos de pago	15
5. 3. Moodboard	16
5. 3. 1. Página de inicio	17
5. 3. 2. Modals	18
5. 4. Flujo de la aplicación	19
5. 4. 1. Inicial:	19
5. 4. 2. Final:	20
6. DESARROLLO	21
6. 1. Front-end	21
6. 1. 1. Diseño de la interfaz	22
6. 1. 1. 1. Inicio de sesión	22
6. 1. 1. 2. Registro	23
6. 1. 1. 3. Menú de navegación	24

6. 1. 1. 4. Inicio	24
6. 1. 1. 5. Pie de página	27
6. 1. 1. 6. Películas, series y mi lista	27
6. 1. 1. 7. Reproductor de contenido	28
6. 1. 1. 8. Página de reseñas	28
6. 1. 1. 9. Playlist	29
6. 1. 1. 10. Panel del administrador	31
6. 1. 1. 11. Mi plan	31
6. 1. 1. 12. Configuración	32
6. 1. 1. 13. Aviso legal	32
6. 1. 1. 14. Perfil	33
6. 1. 1. 15. Sobre nosotros	34
6. 1. 2. Modo claro	34
6. 1. 2. 1. Página de inicio	34
6. 1. 2. 2. Vista de películas	35
6. 1. 2. 3. Perfil	36
6. 1. 3. Responsive	37
6.2. Back-end	39
6. 2. 1. Migraciones:	39
6. 2. 2. Modelos:	43
6. 2. 3. Controladores:	45
6. 2. 4. Rutas:	52
6. 2. 4. 1. Peticiones al servidor	54
6. 2. 5. Relaciones N:M	55
6. 2. 6. Funcionalidades extra	57
6. 2. 6. 1. Firebase - Realtime Database	57
6. 2. 6. 2. Tareas de programación	62
6. 2. 7. Thunder client	64
6. 2. 8. Pruebas automatizadas	65
7. INFORMACIÓN ADICIONAL	71
7. 1. Error 429 (Too many requests)	71
7. 1. Gestión de archivos multimedia	72
8. RESULTADO FINAL	73
9. CONCLUSIONES	73
10. WEBGRAFIA	74

1.INTRODUCCIÓN

Como tercer proyecto del año, se llevará a cabo un trabajo grupal con el objetivo de crear una página web mediante el uso de las nuevas tecnologías y lenguajes de programación aprendidos en clase, tales como Laravel o Vue.

Este tendrá una duración de dos o tres meses aproximadamente y constará de varias fases, desde el planteamiento de la idea principal hasta la puesta en marcha de la plataforma web, pasando por la creación del diseño, el prototipo, el desarrollo del código y el testeo.

2.CONTEXTO Y MOTIVACIÓN

2.1. Objetivos generales de producción

El objetivo principal de este proyecto, consiste en realizar la creación de una página o plataforma web orientada al sector audiovisual, donde poder acceder a un gran catálogo de películas tanto de forma gratuita como a través de suscripciones mensuales.

Además, para marcar la diferencia con el resto de la competencia, se podría añadir alguna funcionalidad para poder ver las películas de forma compartida junto con otros usuarios o compartir listas de reproducción por ejemplo, con el objetivo de fomentar el arte audiovisual y mantener la actividad en la plataforma.

2.2. Objetivos de aprendizaje

Con el desarrollo de este proyecto, se busca adquirir, repasar y practicar los siguientes conceptos:

- Aprender y aumentar nuestros conocimientos sobre los distintos lenguajes de programación mencionados este año PHP, JS, VUE, etcétera.
- Aprender la aplicación de conceptos del diseño web.
- Creación de interfaces web
- Aprender a analizar e identificar errores UX/UI.
- Implementación de estilos bootstrap
- Combinación de los distintos lenguajes mencionados anteriormente
- Creación de un código limpio, organizado y bien estructurado
- Aprender y mejorar en el desarrollo de aplicaciones web mediante el framework de Laravel
- Realizar con éxito un proyecto web funcional y apto para su uso.

3. DESCRIPCIÓN

3.1. Resultados esperados

Respecto a los resultados finales, se espera llevar a cabo la creación de una aplicación web totalmente funcional, que permita a todos los usuarios disfrutar de sus películas favoritas y durante cualquier momento del día, tanto en solitario como acompañado.

Por otro lado, se busca aplicar cada una de las ideas necesarias para el buen funcionamiento de esta, como por ejemplo la implementación de un sistema de suscripciones, el sistema de filtrado de la información y creación de playlist, y añadir métodos para almacenar cierta información como la de la cuenta del usuario o las películas favoritas.

Todo mediante el uso de un código ordenado y bien estructurado, junto con una buena gestión de las bases de datos necesarias para almacenar toda la información.

3.2. Beneficios

Los beneficios que se esperan obtener con este proyecto, son:

- Acceso a contenido variado para los usuarios
- Accesibilidad para permitir a los usuarios ver películas en cualquier momento y lugar
- Calidad de transmisión
- Compatibilidad multiplataforma
- Interfaz intuitiva y fácil de usar para mejorar la usabilidad
- Permitir a los usuarios acceder a un amplio catálogo de contenido mediante una suscripción mensual
- Funciones variadas para mejorar la experiencia de usuario

4. ANÁLISIS

4.1. Requisitos funcionales

Como se ha mencionado anteriormente, este proyecto está pensado para proporcionar material audiovisual a todos aquellos usuarios que se suscriban a la plataforma.

En cuanto a los requisitos funcionales, destacan:

- RF01: Inicio de sesión
- RF02 Registro de los usuarios
- RF03: Sistema de suscripciones mensuales
- RF04: Ver las películas compartidas en la web y disponibles para el plan de suscripción del usuario.
- RF05: Añadir reseñas y puntuaciones sobre las películas, con el objetivo de ayudar a los futuros usuarios.
- RF06: Poder guardar todas aquellas películas que más nos gusten, para poder verlas siempre que nos apetezca.
- RF07: Proporcionar cierta información sobre la película a todos aquellos usuarios que lo deseen.

4.2. Requisitos no funcionales

- RNF01: Evitar los tiempos de carga excesivamente largos, para no aborrecer al usuario.
- RNF02: Mostrar los archivos multimedia con la máxima calidad posible, para no confundir al usuario y generar confusiones.
- RNF03: No almacenar la información sensible de los usuarios como texto plano.
- RNF04: Hacer un buen uso de todo lo relacionado con el UX/UI, para mejorar la experiencia de los usuarios al utilizar nuestra plataforma web.
- RNF05: No utilizar elementos que puedan dificultar o entorpecer la accesibilidad al sitio web, como por ejemplo, mediante menús incoherentes.

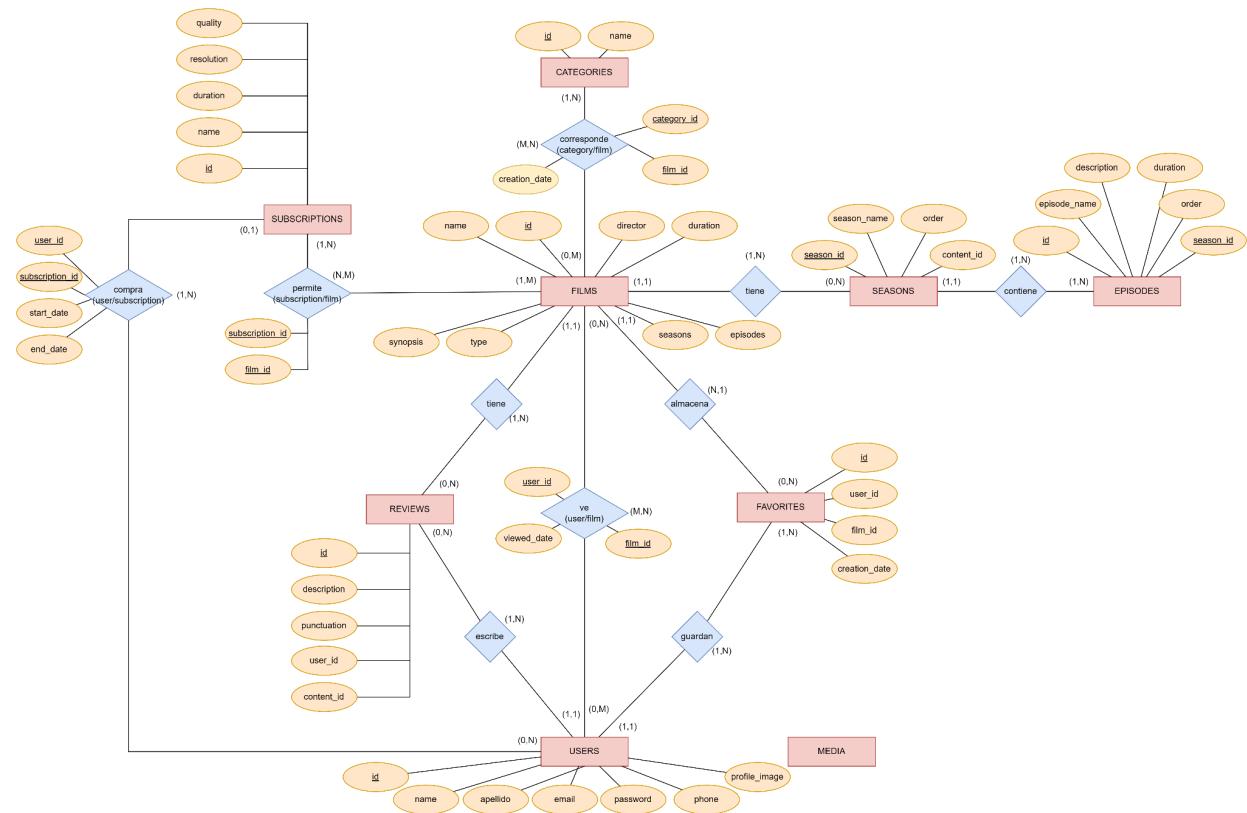
4.3. Requisitos de la base de datos

En cuanto al sistema de los datos, nuestra página web deberá gestionar cierta información de interés como por ejemplo:

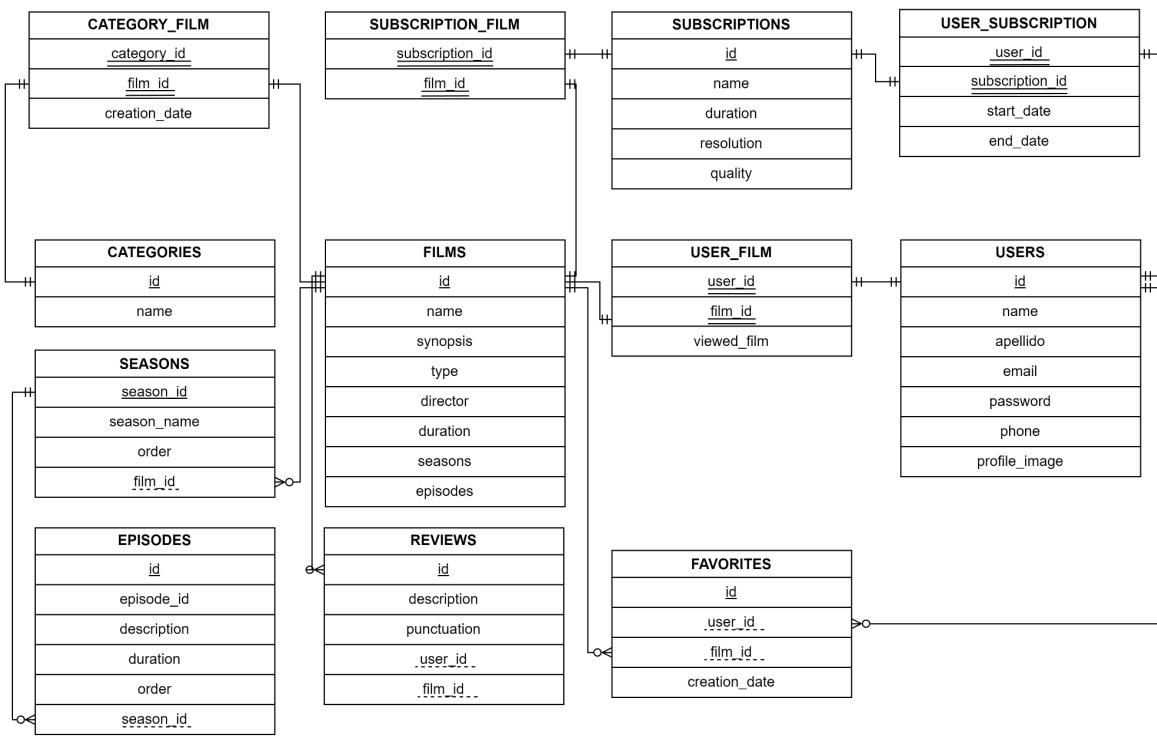
1. **USUARIOS:** Identificador del usuario, nombre, apellido, correo electrónico, contraseña, número de teléfono.
2. **PLANES DE SUSCRIPCIÓN:** Identificador del plan de suscripción, nombre del plan, precio, duración, resolución y calidad.
3. **CATEGORÍAS:** Identificador de la categoría, nombre de la categoría.
4. **CONTENIDO** (tanto películas como series): Identificador de la película, nombre, sinopsis, director, duración, episodios, sesiones, tipo.
5. **TEMPORADAS:** Identificador de la temporada, nombre y orden.
6. **EPISODIOS:** Identificador del episodio, nombre, duración, sinopsis y orden.
7. **RESEÑAS:** Identificador de la reseña, descripción y puntuación.
8. **FAVORITOS:** Identificador tanto del usuario, como del contenido que decide almacenar.

4.3. 1. Diseño de la base de datos

4. 3. 1. 1. Modelo entidad relación



4. 3. 1. 2. Modelo relacional



5. DISEÑO

5.1. Definición de la empresa e identificación del target

Esta empresa o proyecto web, está destinada a proporcionar información acerca de todas aquellas películas que se vayan publicando, además de intentar hacer pasar un buen rato a los posibles clientes.

En cuanto al target o público objetivo, este abarca un baremo muy grande de edades, teniendo en cuenta la gran cantidad de géneros que existen. Por ello, el rango de edad girará en torno a los siete y sesenta y cinco años.

Mencionar que todo el contenido de la página web estará estrictamente categorizado, con el objetivo de proporcionar el mejor contenido posible a cada rango de edad y de la forma más acertada posible.

5. 1. 1. Identificación de la imagen corporativa de la empresa

La imagen corporativa de iWatch está formada principalmente por un logotipo, compuesto por el mismo nombre de la empresa y cuya tipografía sans serif se ha diseñado a mano, con el objetivo de aportar un toque más personal y con cierto sentimiento de modernidad y tecnología.

Este logotipo, como el rostro de su marca, sugiere profesionalismo y originalidad, añadiendo el toque tecnológico, moderno y de accesibilidad, mencionados anteriormente.

Mediante este conjunto, logramos un conjunto que transmite la idea de un proyecto innovador, adaptando las tendencias actuales y que esta diseñado para brindar una experiencia de usuario fluida.

The logo consists of the word "iWATCH" in a bold, black, sans-serif font. The letters are all uppercase and have a thick, rounded appearance. There is a small gap between the "i" and the "W".

5. 2. Wireframe

Para definir la estructura básica de nuestra página web, hemos realizado un wireframe. De esta forma podremos trazar el diseño y la disposición de cada uno de los elementos de la plataforma, de forma sencilla y coherente.

Mencionar que esto es un esbozo inicial, durante el transcurso del proyecto se han ido modificando ciertos aspectos con el objetivo de alcanzar los requisitos y desarrollar la página web de la mejor forma posible.

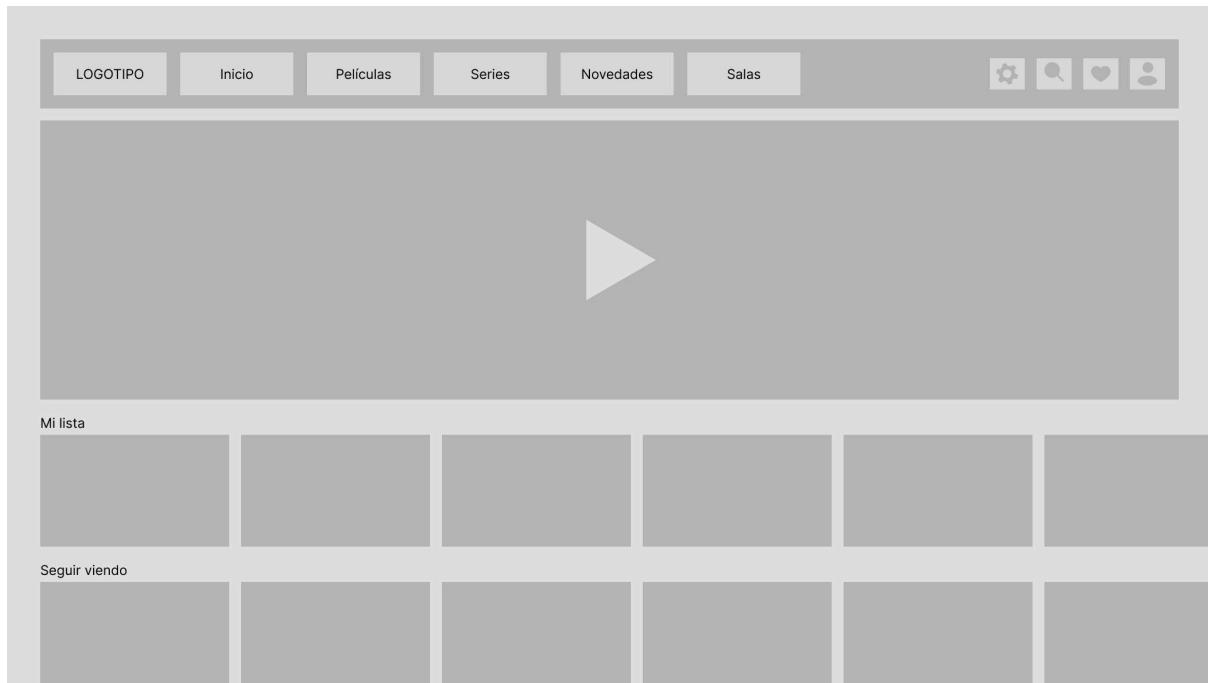
5. 2. 1. Registro de sesión

The wireframe shows a registration form titled 'REGISTRO'. It consists of six input fields: 'Nombre', 'Apellido', 'Correo electrónico', 'Contraseña', 'Confirma la contraseña', and 'Número de teléfono'. Below these fields is a checkbox labeled 'Acepta los términos y condiciones'. At the bottom is a large button labeled 'REGISTRARSE'.

5. 2. 2. Inicio de sesión

The wireframe shows a login form titled 'INICIAR SESIÓN'. It has two input fields: 'Correo electrónico' and 'Contraseña'. Below these is a large button labeled 'INICIAR SESIÓN'. At the bottom left is a checkbox labeled 'Recuérdame', and at the bottom right is a link labeled '¿Necesitas ayuda?'.

5. 2. 3. Sala principal



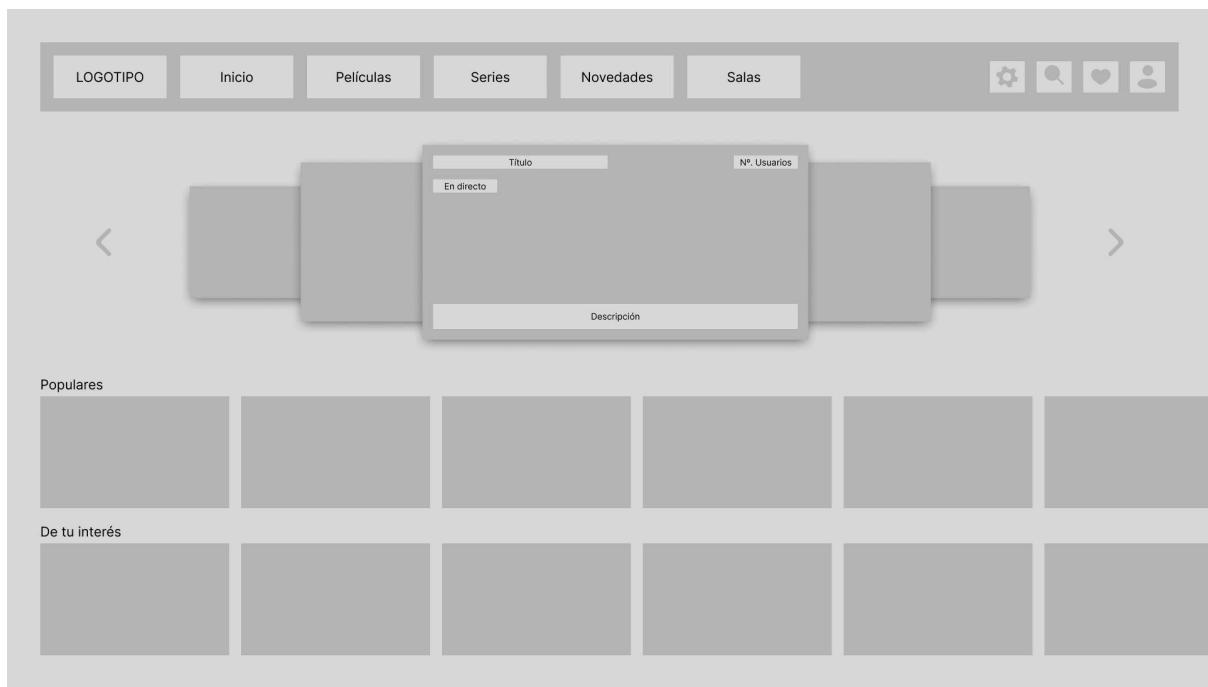
5. 2. 4. Información de la película



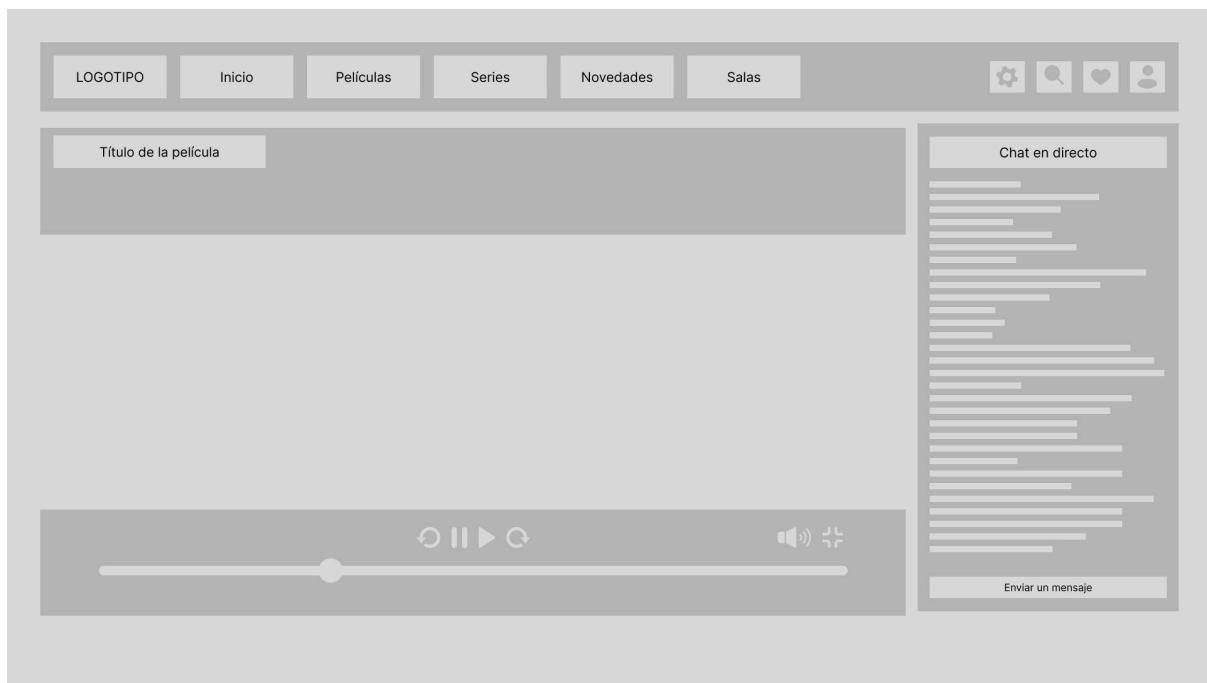
5. 2. 5. Reproductor



5. 2. 6. Salas de cine



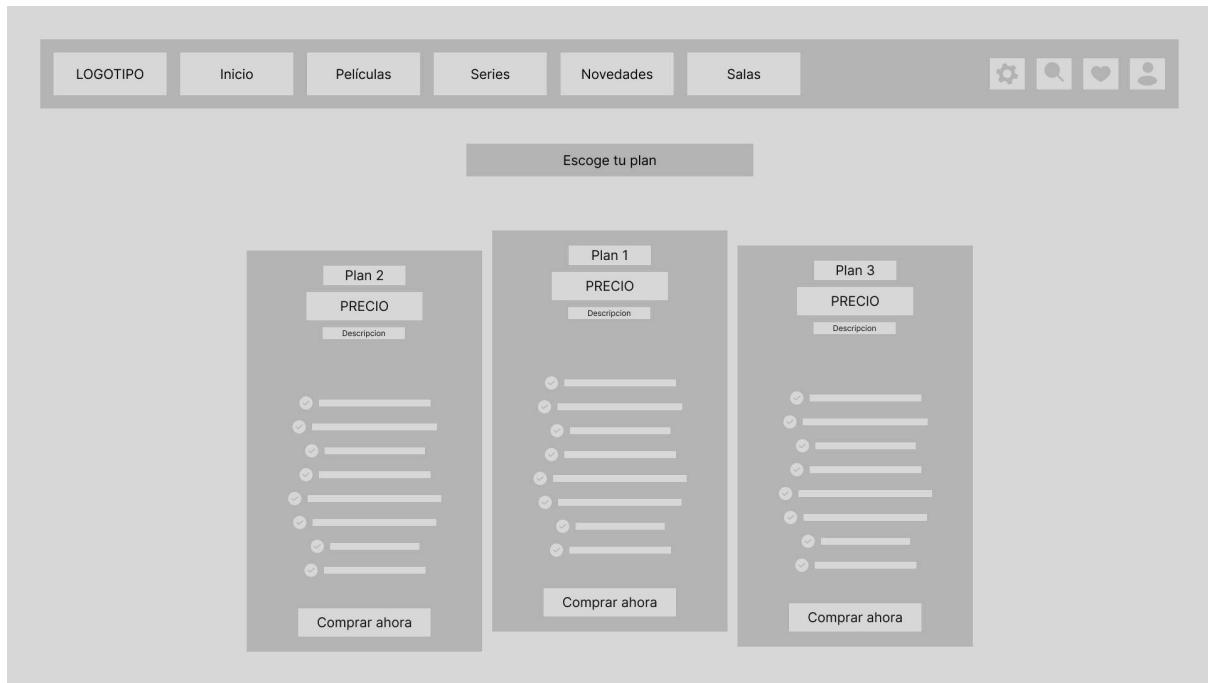
5. 2. 7. Sala



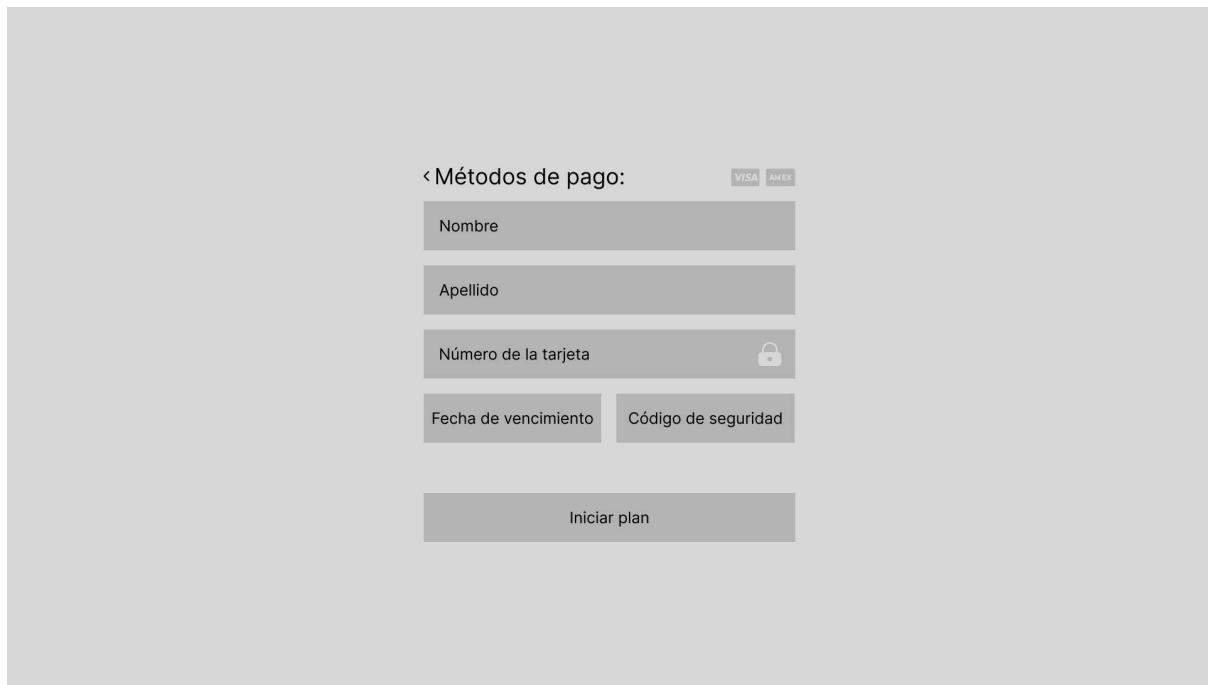
5. 2. 8. Perfil



5. 2. 9. Plan de suscripciones



5. 2. 10. Métodos de pago



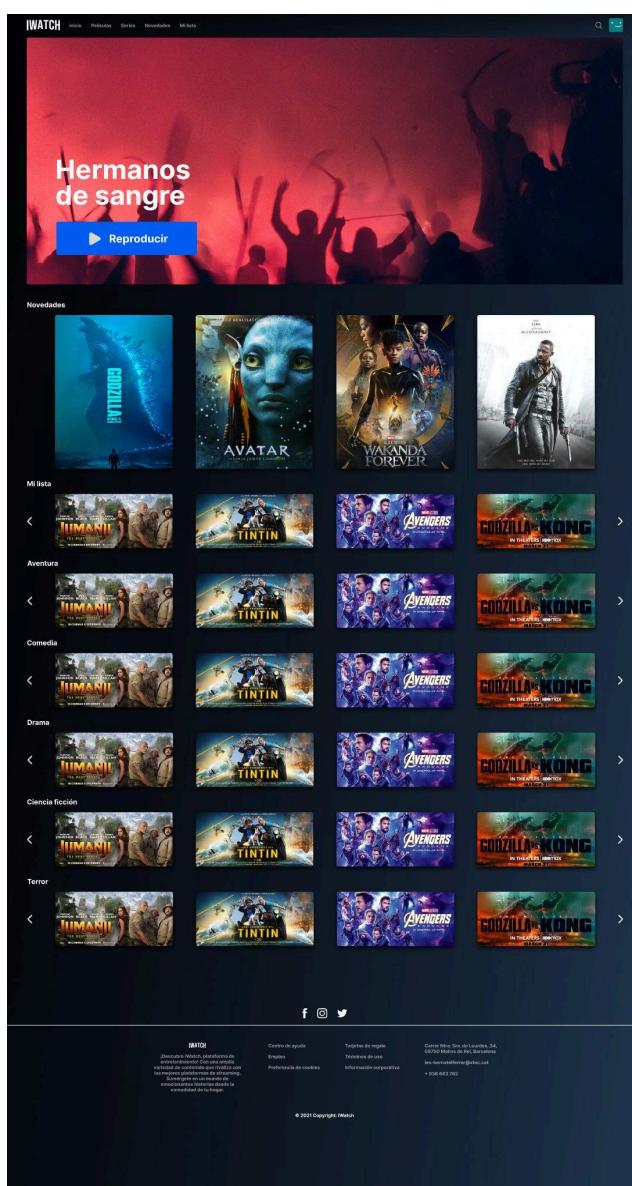
Enlace:<https://www.figma.com/file/3PCbwNF6RYr9ZULKTRh2kJ/LARAVEL?type=design&node-id=41%3A211&mode=design&t=J4fcZCnSb0f1Y0oD-1>

5. 3. Moodboard

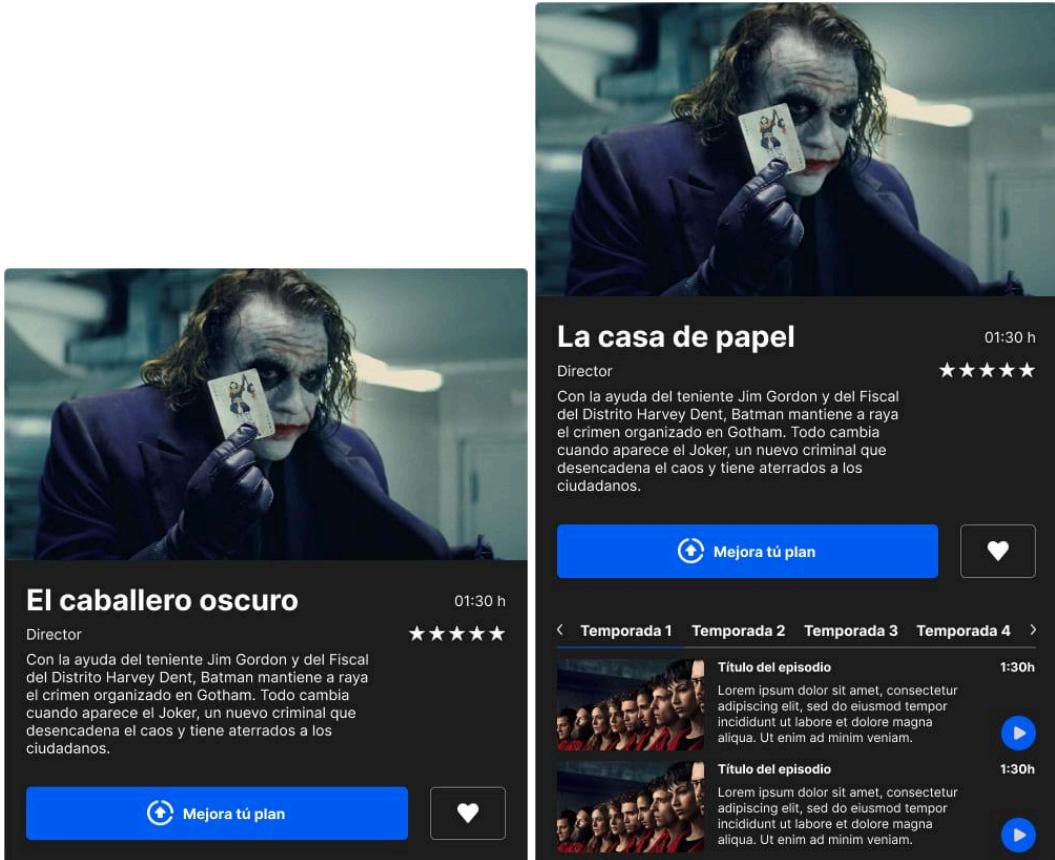
Con el objetivo de crear una imagen o idea previa al proyecto, decidimos llevar a cabo la creación del moodboard, el cual encapsula la esencia visual y la creatividad que se busca transmitir. Este se encarga de fusionar diferentes aspectos de gran importancia como por ejemplo los colores, las tipografías, las imágenes y otros elementos gráficos de gran importancia y que reflejan la identidad y el propósito de nuestro proyecto.

En este caso, nos enfocamos en algunas de las ventanas principales de la página web, como por ejemplo la página de inicio y los modals o que muestran la información de las películas:

5. 3. 1. Página de inicio

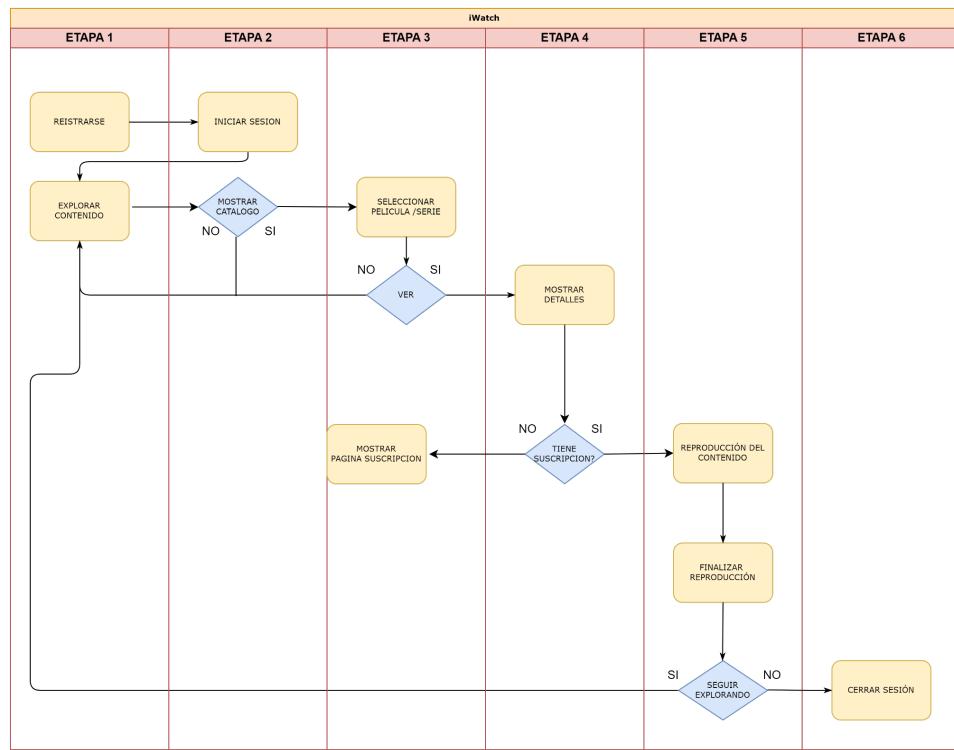


5. 3. 2. Modals

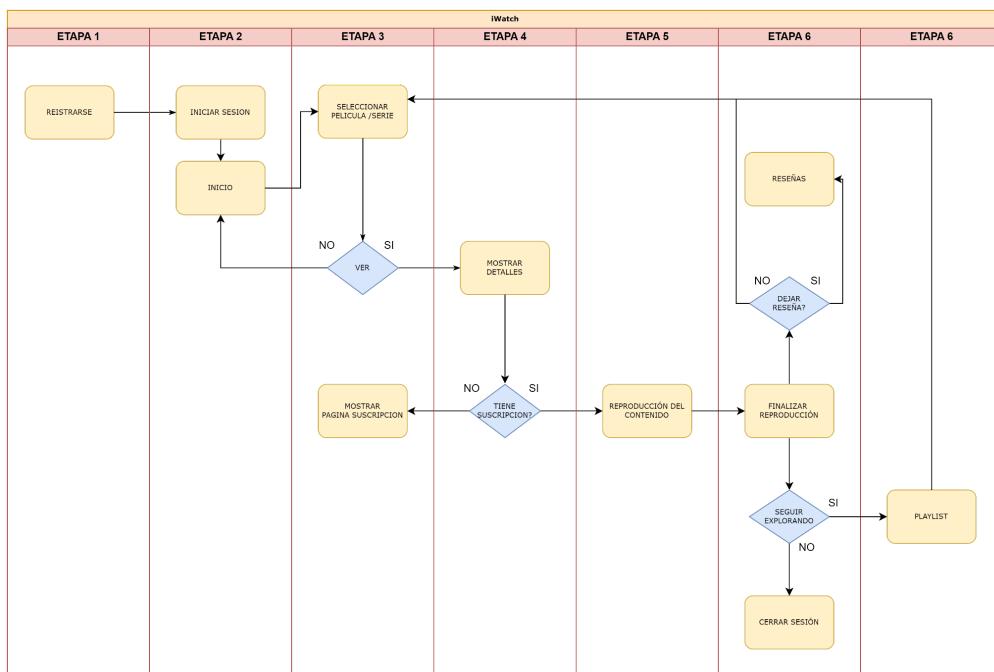


5. 4. Flujo de la aplicación

5. 4. 1. Inicial:



5. 4. 2. Final:



6. DESARROLLO

Tras definir la idea principal del proyecto, realizar el desarrollo de los esbozos iniciales, la creación tanto del moodboard como del wireframe y definir el flujo de la aplicación, toca empezar con el desarrollo de la página web.

Para ello, utilizaremos dos frameworks muy conocidos: Laravel 10 para el tema del back-end y Vue 3 de JavaScript, enfocado principalmente al desarrollo de interfaces de usuario.

6. 1. Front-end

Como se ha mencionado anteriormente, en este proyecto utilizaremos los recursos proporcionados por Vue 3, con el objetivo de crear una página organizada, con una buena estructura y modularización.

Además, añadiremos componentes de vue, bloques de construcción reutilizables que encapsulan una parte de la interfaz del usuario junto con su lógica asociada.

Por último, mencionar que a la hora de aplicar estilos, como en los proyectos anteriores, se usará tanto CSS como Bootstrap, con el objetivo de mejorar tanto la experiencia de usuario como la interfaz, aplicando metodologías responsive para poder adaptar nuestra web a cualquier dispositivo.

6. 1. 1. Diseño de la interfaz

Respecto al diseño de nuestra plataforma web, hemos optado por una composición limpia, moderna y no muy cargada para no saturar al usuario.

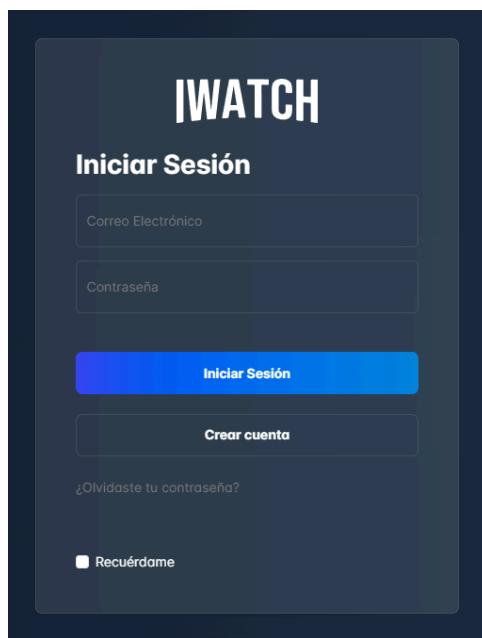
Esta se caracteriza por el uso de formas con acabados redondeados, lo que implica una apariencia mucho más visible y amigable, gracias a las curvas dinámicas y fluidas, las cuales transmiten agilidad e ingenio.

Por otro lado tenemos la tipografía, otro de los aspectos fundamentales de nuestra plataforma. En este caso tenemos la fuente Arial, una tipografía sans serif y de palo seco, que aporta una gran legibilidad en pantallas y a su vez, transmite esa esencia de modernismo o temática relacionada con todo lo tecnológico.

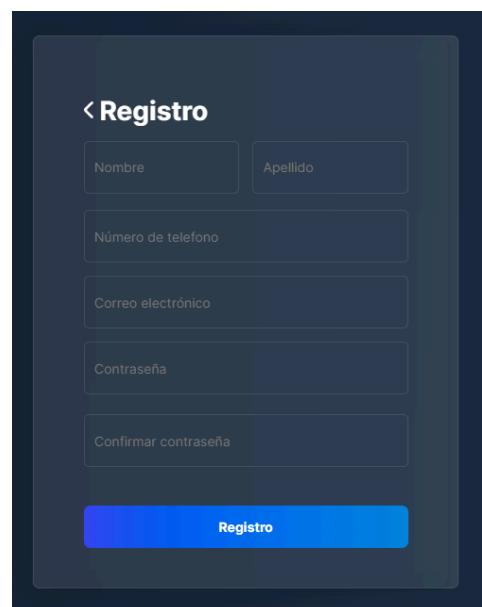
Y por último, mencionar el color, en cuyo caso nos decantamos por una gama cromática principalmente oscura con el objetivo de mantener un ambiente cinematográfico, ya que evoca el sentimiento de una sala de cine oscura y, además, permite diferenciar el contenido principal del fondo de la web, ya que la mayoría de las carátulas, suelen tener colores llamativos que ayudan a llamar la atención del usuario o del cliente.

Destacar que en algunos casos, hemos añadido ciertas animaciones como por ejemplo en el caso de los carruseles de la página inicial, los cuales se activan mediante el desplazamiento de la página. Una funcionalidad sencilla pero que aporta cierto dinamismo a la página.

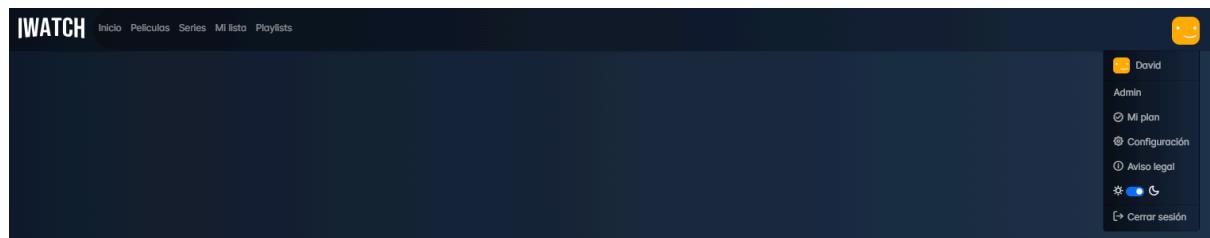
6. 1. 1. 1. Inicio de sesión



6. 1. 1. 2. Registro



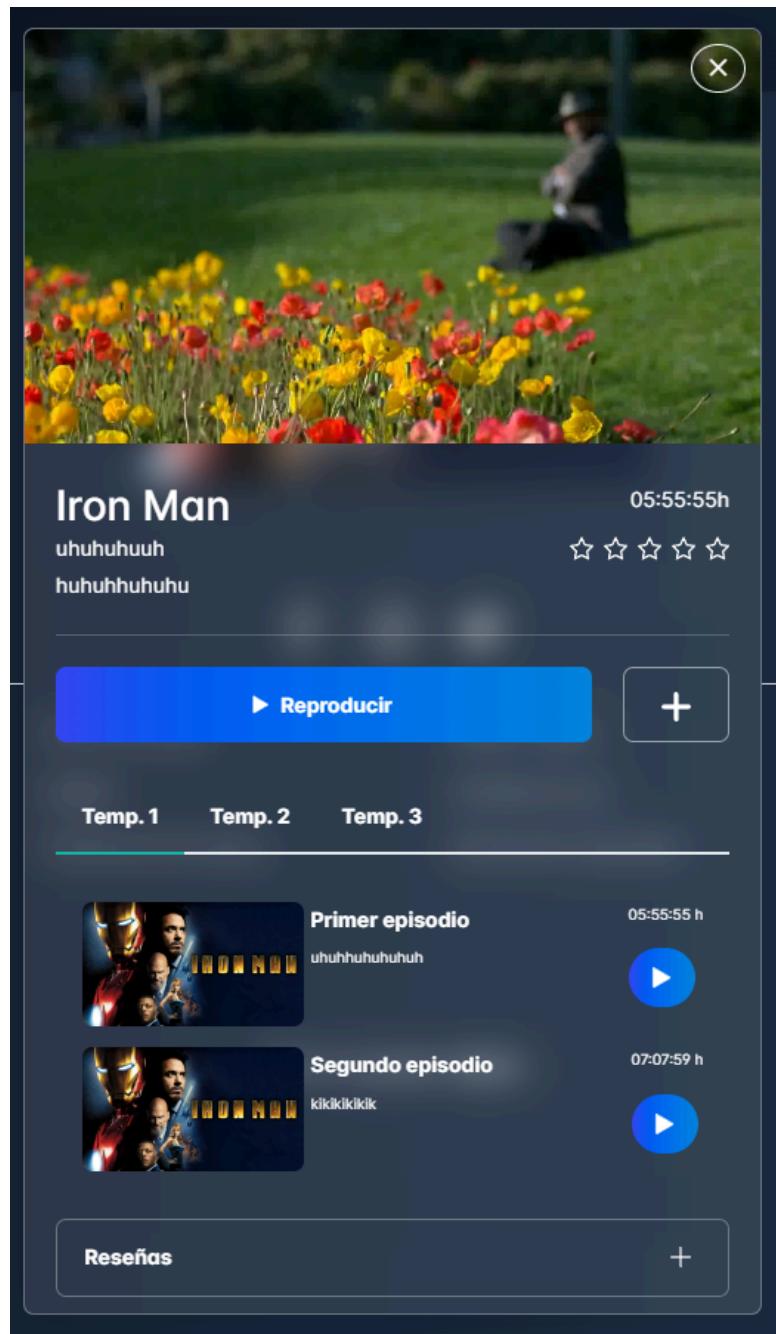
6. 1. 1. 3. Menú de navegación

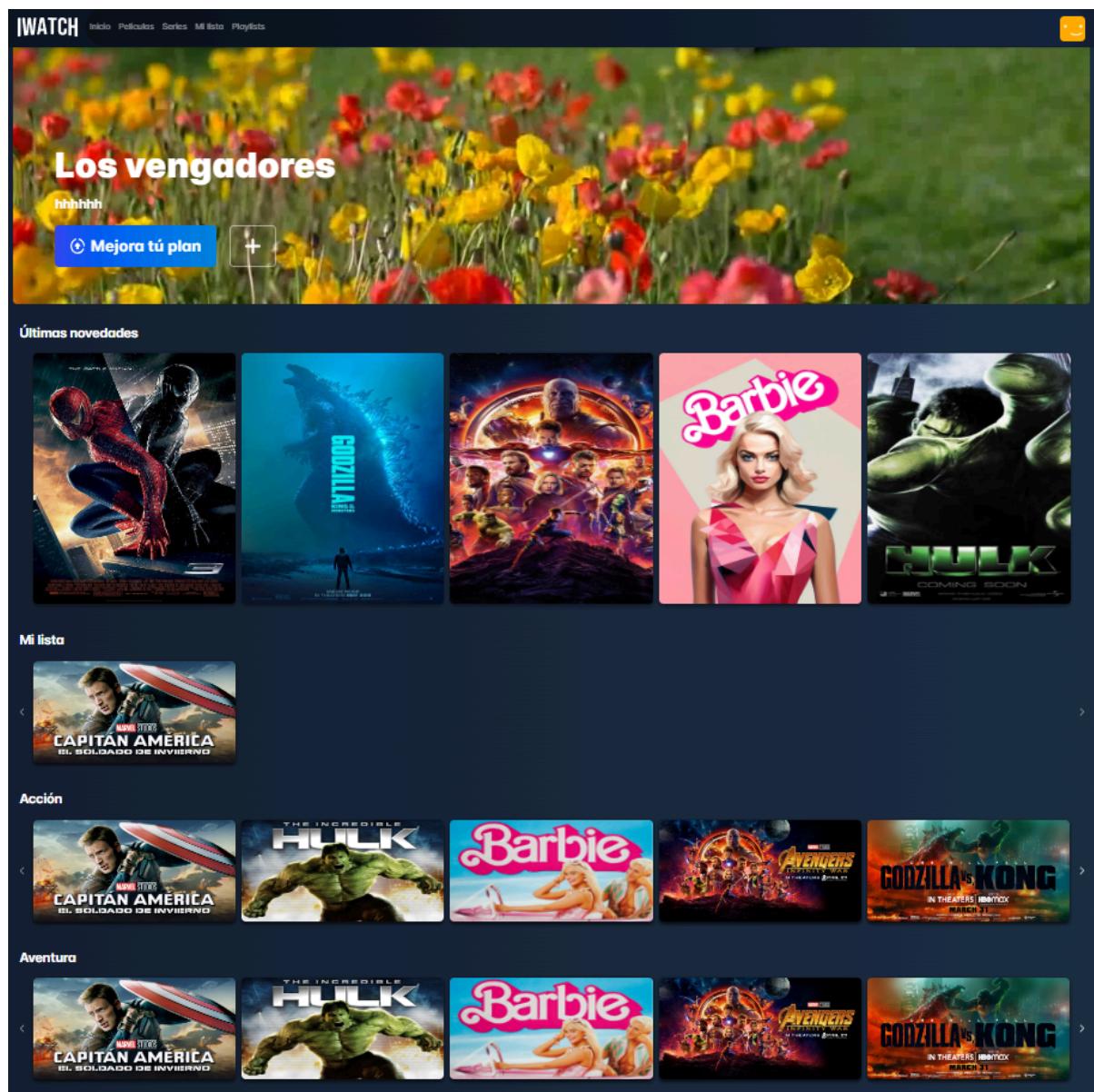


6. 1. 1. 4. Inicio

The main page of the iWATCH web application. It features a large banner for the movie 'Capitan America' with a background image of a person sitting in a field of flowers. Below the banner, there is a call-to-action button labeled 'Mejora tú plan' and a '+' sign. A section titled 'Últimas novedades' displays five movie posters: 'Avengers: Infinity War', 'Spider-Man', 'Dwayne Johnson', 'Mel Gibson', and another 'Avengers: Infinity War' poster.

A detailed view of the 'Spider-Man' movie card from the iWATCH web application. The card shows the movie title 'Spider-man' (Sam Raimi, El hombre araña), a duration of '01:55:59h', and a rating of five stars. It includes a 'Reproducir' (Play) button, a '+' sign, and a feedback section asking '¿Que te ha parecido?'. The card is overlaid on a larger image of the movie's promotional poster.





6.1.1.5. Pie de página



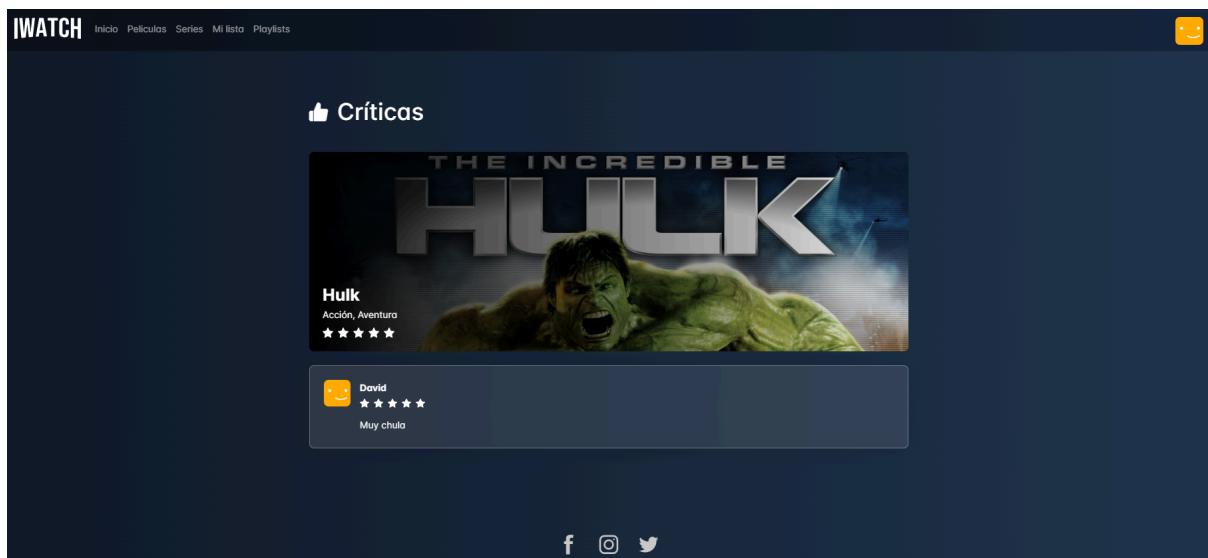
6. 1. 1. 6. Películas, series y mi lista

The screenshot shows the iWATCH website interface. At the top, there's a navigation bar with links for 'Inicio', 'Películas', 'Series', 'Mi lista', and 'Playlists'. A search bar says 'Introduce el título...'. On the right, there's a dropdown menu for genres: 'Mostrar todo' (Show all), 'Acción' (Action), 'Aventura' (Adventure), 'Comedia' (Comedy), and 'Drama' (Drama). Below the search bar, there's a section titled 'Películas' (Movies) with five movie posters: 'Capitán América: El Soldado de Invierno', 'The Incredible Hulk', 'Barbie', 'Avengers: Infinity War', and 'Godzilla vs. Kong'. At the bottom of the page, there's a footer with social media icons for Facebook, Instagram, and Twitter, followed by a detailed contact section.

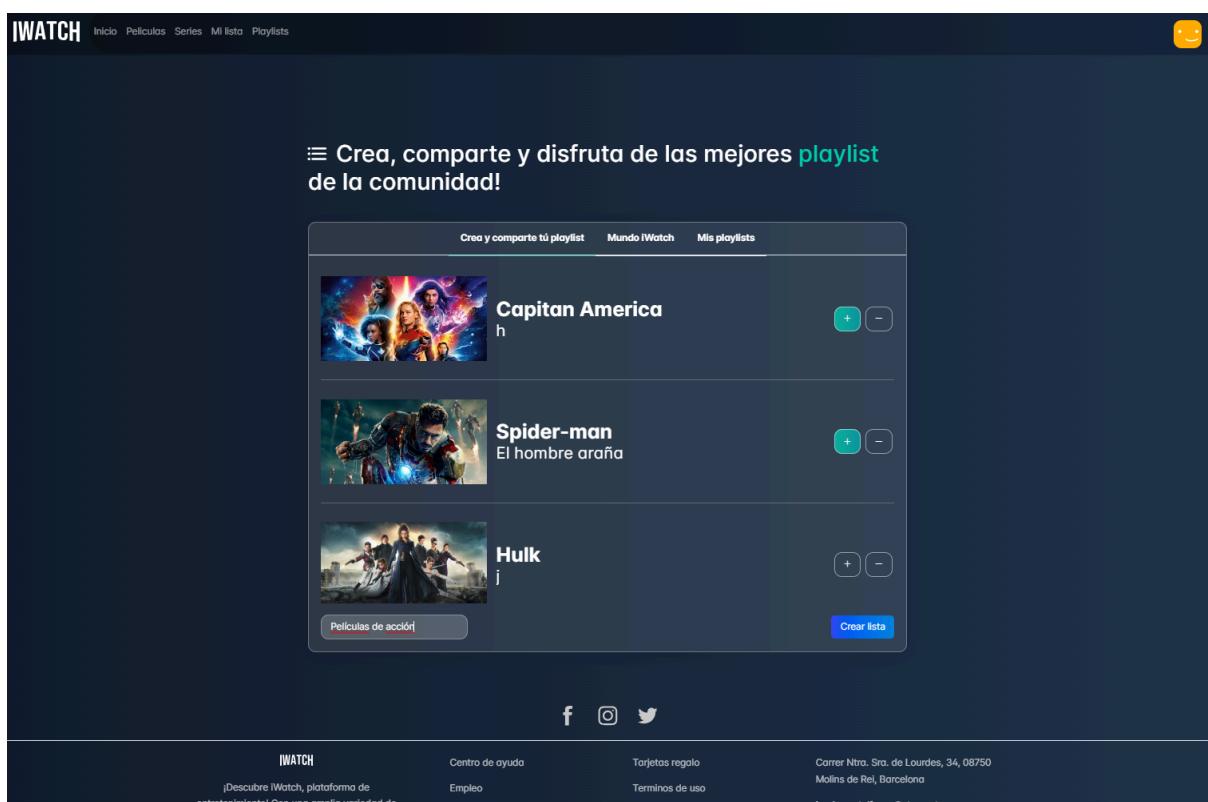
6. 1. 1. 7. Reproductor de contenido



6. 1. 1. 8. Página de reseñas



6. 1. 1. 9. Playlist



Desarrollo de aplicaciones web

The screenshot shows the iWatch website with a dark blue header. The top navigation bar includes links for 'Inicio', 'Películas', 'Series', 'Mi lista', and 'Playlists'. On the far right of the header is a yellow square icon with a smiley face. Below the header, a large teal banner features the text 'Crea, comparte y disfruta de las mejores playlist de la comunidad!' with a small icon of three dots and a line. The main content area has a light gray background and contains a large rounded rectangle containing a list of movies. At the top of this list is a card for 'Mejores películas 2021' featuring 'Hulk' and 'Spider-man: El hombre araña'. Below this are two more cards: 'Películas de acción' and 'Películas de terror!!'. Each card has a heart icon with a plus sign on its right side. At the bottom of the page are social media links for Facebook, Instagram, and Twitter, followed by a footer with links for 'iWATCH', 'Centro de ayuda', 'Tarjetas regalo', 'Carrer Ntra. Sra. de Lourdes, 34, 08750 Molins de Rei, Barcelona', and other legal information like 'Empleo', 'Terminos de uso', and 'Preferencia de cookies'.

This screenshot shows the same iWatch website but with a light blue header. The layout and content are identical to the dark version, including the navigation bar, teal banner, movie list, and footer. The only visual difference is the color scheme, with a light blue header instead of a dark one.

6. 1. 1. 10. Panel del administrador

The screenshot shows the iWatch administrator interface. On the left is a sidebar with navigation links for HOME, USUARIOS, PELÍCULAS (with 'Listar películas' highlighted), SERIES, TEMPORADAS, and EPISODIOS. The main area is titled 'Todas las películas' and displays a table of movie records. Each row includes columns for #, Nombre, Sinopsis, Director, Puntuación, Duración, Episodios, Temporadas, Categoría, Tipo, suscripción, Poster, Poster h, Video, and Acciones (with 'Editor' and 'Eliminar' buttons). The table lists six movies: Capitán América, Hulk, Barbie, Los vengadores, Godzilla, and Spider-man.

6. 1. 1. 11. Mi plan

The screenshot shows the iWatch subscription plan comparison page. At the top, it says 'Escoge el mejor plan para ti' and encourages users to choose a plan that fits their preferences. Below this, three plans are compared: Básico (9.95€), Estándar (14.95€), and Premium (19.95€). Each plan has a description and an 'Upgrade plan' button. The Premium plan is highlighted with a teal background. At the bottom, it says 'Compara nuestros planes y encuentra el que más se adapte a ti' and provides a table comparing the three plans based on Nombre, Precio, Duración, Resolución, and Calidad.

Nombre	Precio	Duración	Resolución	Calidad
Básico	9.95	06:33:14	SD	Baja
Estándar	14.95	06:33:14	HD	Media
Premium	19.95	06:33:14	4K	Alta

6. 1. 1. 12. Configuración

The screenshot shows the configuration menu of the iWatch application. The main title is "Configuración". The menu is organized into several sections:

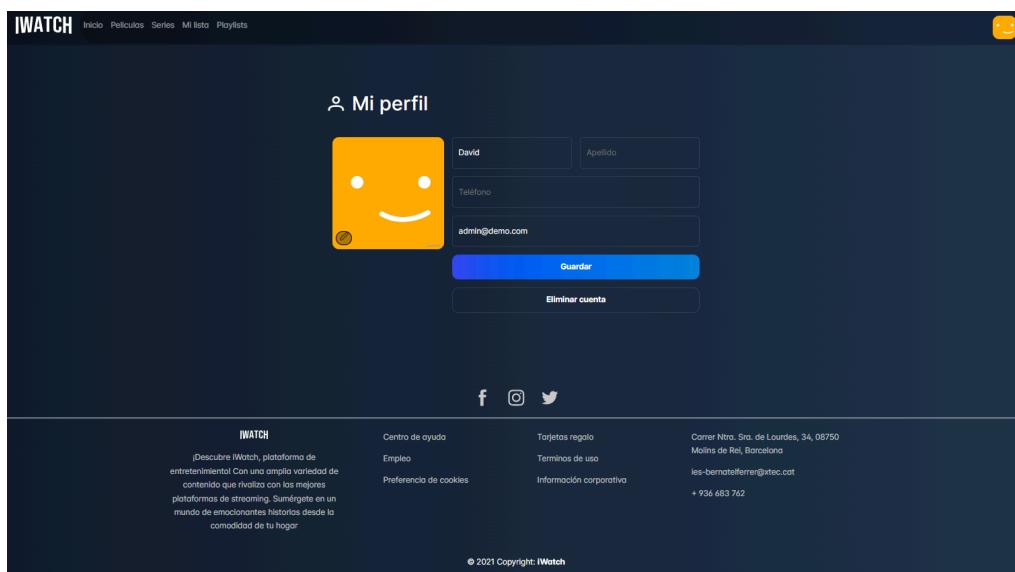
- CUENTA**: Includes options like Actualizar información personal, Cambiar contraseña, Mi plan, Control parental, Configuración de dispositivos, and Historial de visualización.
- REPRODUCCIÓN**: Includes options like Calidad de reproducción, Configuración de subtítulos y audio, Configuración de calidad de audio, and Configuración de calidad de video.
- PREFERENCIAS DE USUARIO**: Includes options like Reproducción automática, Idioma y subtítulos predeterminados (with a dropdown menu showing "ES").
- DESCARGAS Y ALMACENAMIENTO**: Includes options like Descargas, Configuración de espacio de almacenamiento, and Gestión de descargas activas.
- NOTIFICACIONES**: Includes the option Configuración de notificaciones por correo electrónico.

6. 1. 1. 13. Aviso legal

The screenshot shows the legal notice page of the iWatch application. The main title is "Aviso legal". It includes the following sections:

- Última Actualización: 20-04-2024**
- Términos y condiciones**: States that by accepting and using the services, the user agrees to comply with the terms and conditions. It also states that if the user disagrees with any part of the terms, they should not use the services.
- Condiciones de uso**: Lists rules such as not interfering with platform operation, not attempting unauthorized access, and not using services for illegal purposes.
- Contenidos**: States that contents are protected by intellectual property law and belong to iWatch or its licensors.
- Responsabilidad**: States that iWatch will not be responsible for damages resulting from the use of services.
- Modificación de términos**: States that iWatch reserves the right to modify terms and conditions at any time, and continued use after modification constitutes acceptance.
- Última Actualización: 20-04-2024**
- Política de Privacidad**: States that iWatch collects personal information from users, such as name, email, date of birth, gender, preferences, and viewing history, to provide personalized experiences.
- Información personal**: Details what personal information is collected, including account creation and interaction data.
- Reservarios**: States that iWatch uses various means to collect personal information.

6. 1. 1. 14. Perfil



6. 1. 1. 15. Sobre nosotros

¡Bienvenido a iWatch!

En iWatch, nos apasiona el entretenimiento y queremos brindarte la mejor experiencia para disfrutar de tus películas y series favoritas. Con una amplia selección de películas, series originales, reseñas de contenido y mucho más, iWatch es tu destino principal para el entretenimiento en línea. Descubre lo que nos hace únicos y únete a nuestra comunidad de amantes del cine y la televisión.

En iWatch, nos esforzamos por ofrecerte una plataforma de streaming excepcional que satisface todas tus necesidades de entretenimiento. Desde el lanzamiento de nuestra plataforma, nos hemos dedicado a proporcionar un acceso fácil y conveniente a una gran variedad de contenido, desde éxitos de taquilla hasta emocionantes series originales. Nuestro equipo está compuesto por apasionados del cine y la tecnología, comprometidos a mejorar continuamente tu experiencia de visualización. Nos enorgullece ofrecer una interfaz intuitiva, funciones personalizadas y una atención al cliente excepcional para garantizar que tu tiempo en iWatch sea siempre satisfactorio. En iWatch, no solo se trata de ver películas y series, sino de sumergirse en un mundo de historias emocionantes y momentos inolvidables. Únete a nosotros y descubre el poder del entretenimiento en línea en iWatch.

Nuestros servicios

En iWatch, ofrecemos una amplia gama de servicios para satisfacer tus necesidades de entretenimiento:

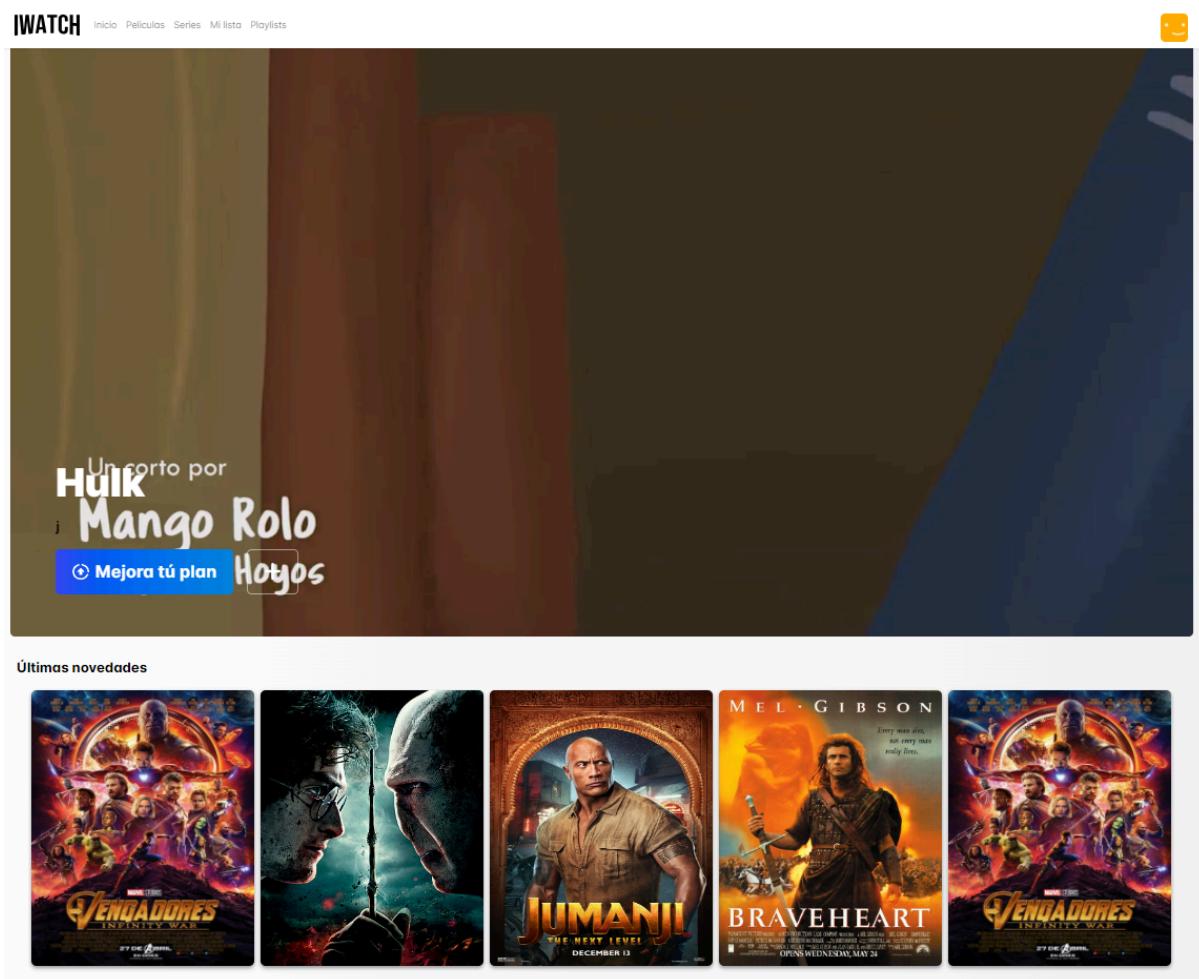
- **Suscripciones:** Accede a una variedad de planes de suscripción flexibles para adaptarse a tu estilo de vida y presupuesto.
- **Películas y Series:** Explora nuestra extensa biblioteca de películas y series, que se actualiza regularmente con los últimos lanzamientos y clásicos temporales.
- Obtén información detallada sobre las películas y series disponibles en nuestra plataforma, con reseñas exhaustivas para ayudarte a tomar decisiones informadas sobre qué ver a continuación.

6. 1. 2. Modo claro

La capacidad de cambiar el tema de una aplicación, no es solo una característica estética, sino una herramienta que influye en la experiencia del usuario de múltiples maneras.

Por ello, quisimos aplicar dicha funcionalidad a nuestra plataforma, con la finalidad de añadir la posibilidad de adaptar el aspecto visual y que se ajustase a las preferencias de cada usuario, a continuación se muestran algunos ejemplos:

6. 1. 2. 1. Página de inicio



6. 1. 2. 2. Vista de películas

The screenshot shows the 'Películas' (Movies) section of the iWatch app. At the top, there is a search bar with the placeholder 'Introduce el título...' and a dropdown menu labeled 'Mostar todo'. A yellow smiley face icon is in the top right corner. Below the search bar, there is a grid of movie posters. One poster for 'Avengers: Endgame' is highlighted with a larger preview below it. At the bottom, there are social media sharing icons for Facebook, Instagram, and Twitter.

Películas

Introduce el título...

Mostar todo

Smiley icon

iWATCH Inicio Películas Series Mi lista Playlists

Películas

Introduce el título...

Mostar todo

Smiley icon

iWATCH Inicio Películas Series Mi lista Playlists

Centro de ayuda

Tarjetas regalo

Carrer Ntra. Sra. de Lourdes, 34, 08750

Empleo

Terminos de uso

Molins de Rei, Barcelona

f i t

6. 1. 2. 3. Perfil

The screenshot shows the 'Mi perfil' (My Profile) section of the iWatch app. It features a large yellow smiley face icon. To its right are input fields for 'Nombre' (Name) containing 'David', 'Apellido' (Last Name) containing 'Apellido', 'Teléfono' (Phone), and an email address 'admin@demo.com'. Below these fields is a blue 'Guardar' (Save) button. Further down is a white button labeled 'Eliminar cuenta' (Delete account). At the bottom, there are social media sharing icons for Facebook, Instagram, and Twitter.

iWATCH Inicio Películas Series Mi lista Playlists

Smiley icon

Mi perfil

Nombre: David

Apellido: Apellido

Teléfono:

admin@demo.com

Guardar

Eliminar cuenta

f i t

iWATCH Inicio Películas Series Mi lista Playlists

Centro de ayuda

Tarjetas regalo

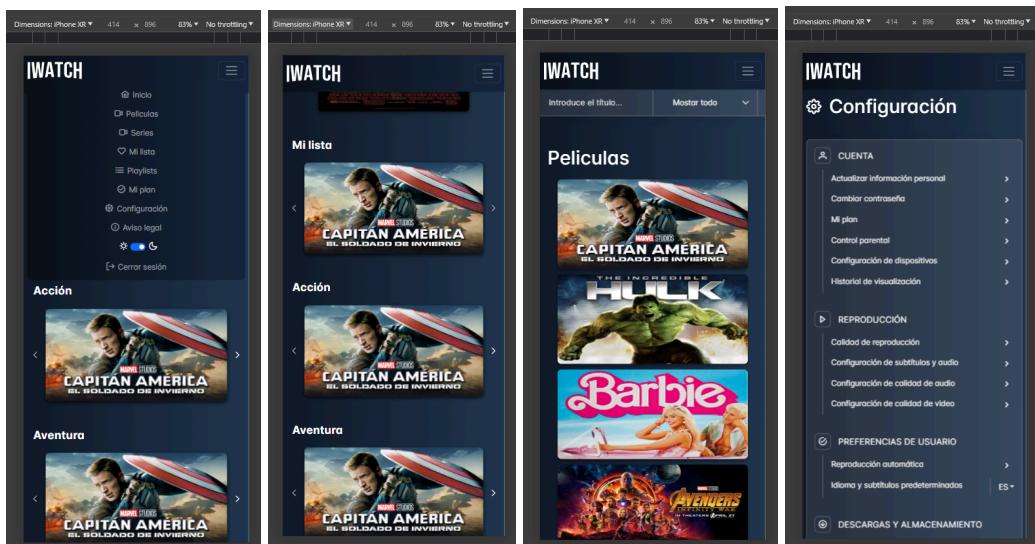
Carrer Ntra. Sra. de Lourdes, 34, 08750

6. 1. 3. Responsive

Para cumplir con el requisito de desarrollar una webapp, establecido al principio del proyecto, aplicamos ciertos sistemas de diseño para poder adaptar la plataforma a cualquier tipo de dispositivo.

En nuestro caso, hemos hecho uso tanto de las clases de bootstrap como de los media queries de CSS y los componentes responsive de Prime Vue.

A continuación, se muestran algunos ejemplos:



6.2. Back-end

Una de las principales características de Laravel, es su automatización a la hora de realizar ciertas tareas o programar determinadas funcionalidades. Además, cuenta con una gran variedad de comandos que nos permitirán agilizar el desarrollo y mejorar tanto la organización como la gestión de los archivos.

6. 2. 1. Migraciones:

En primer lugar y tras realizar un pequeño estudio sobre los datos que necesitábamos almacenar y en qué forma almacenarlos, decidimos elaborar las migraciones, herramientas de bases de datos que permiten modificar su estructura de forma programática.

Estas son una forma de definir los cambios en la base de datos de una manera que se pueda controlar y seguir.

COMANDO: `php artisan make:migration nombre_de_la_migracion`

En nuestro caso, la mayoría de las migraciones disponen de la siguiente estructura:

```

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('films', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('synopsis');
            $table->string('director');
            $table->time('duration');
            $table->string('episodes');
            $table->string('seasons');
            $table->string('type');
            $table->timestamps();
        });

        // Creamos la tabla pivot para la relación muchos a muchos
        Schema::create('category_film', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('category_id');
            $table->unsignedBigInteger('film_id');
            $table->foreign('category_id')->references('id')->on('categories')->onDelete('cascade');
            $table->foreign('film_id')->references('id')->on('films')->onDelete('cascade');
            $table->timestamps();
        });

        Schema::create('suscripcion_film', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('suscripcion_id');
            $table->unsignedBigInteger('film_id');
            $table->foreign('suscripcion_id')->references('id')->on('suscripciones')->onDelete('cascade');
            $table->foreign('film_id')->references('id')->on('films')->onDelete('cascade');
            $table->timestamps();
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('user_film');
        Schema::dropIfExists('category_film');
        Schema::dropIfExists('films');
    }
};

```

Tal y como se muestra, al ejecutar la función UP, se realizarán los comandos de CREATE, los cuales se encargarán de poblar las base de datos con las tablas que especifiquemos. Además, en cada una de ellas, deberemos establecer los campos con sus respectivos tipos de datos y las claves foráneas y primarias, con el objetivo de mantener un orden claro y preciso entre las tablas.

Aparte, también se nos creará la función DOWN, que nos permitirá eliminar todas las tablas que insertemos en dicha función siempre y cuando ejecutemos el comando necesario para ello.

Otros ejemplos:

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('favorites', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id')->nullable();
            $table->unsignedBigInteger('film_id')->nullable();
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            $table->foreign('film_id')->references('id')->on('films')->onDelete('cascade');
            $table->date('creation_date');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('favorites');
    }
};
```

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('apellido')->nullable();
            $table->string('email')->unique();
            $table->timestamp('email_verified_at')->nullable();
            $table->string('password');
            $table->string('phone', 15)->nullable()->unique();
            $table->string('profile_image')->default('/images/Netflix-avatar2.png');
            $table->rememberToken();
            $table->timestamps();
            $table->unsignedBigInteger('suscripcion_id')->nullable();
            $table->foreign('suscripcion_id')->references('id')->on('suscripciones')->onDelete('cascade');
        });

        Schema::create('user_film', function (Blueprint $table) {
            $table->id();
            $table->unsignedBigInteger('user_id');
            $table->unsignedBigInteger('film_id');
            $table->foreign('user_id')->references('id')->on('users')->onDelete('cascade');
            $table->foreign('film_id')->references('id')->on('films')->onDelete('cascade');
            $table->date('viewed_date');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('users');
    }
};
```

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('seasons', function (Blueprint $table) {
            $table->id('season_id');
            $table->string('season_name');
            $table->integer('order');
            $table->unsignedBigInteger('content_id');
            $table->foreign('content_id')->references('id')->onDelete('cascade');
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('seasons');
    }
};
```

6. 2. 2. Modelos:

En laravel, los modelos son las clases que representan a las tablas en la base de datos. Estos se utilizan para interactuar con esta y realizar operaciones CRUD como por ejemplo insertar, actualizar, eliminar y listar datos. Se almacenan en la carpeta app/Models.

Además, nos permiten definir las relaciones entre las tablas/modelos, como por ejemplo las N:M.

COMANDO: php artisan make:model nombre_del_model

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Spatie\ImageManipulations;
use Spatie\MediaLibrary\HasMedia;
use Spatie\MediaLibrary\InteractsWithMedia;
use Spatie\MediaLibrary\Collections\Models\Media;

use App\Models\User;
use App\Models\UserFilm;
use App\Models\Categoría;
use App\Models\Suscripción;

class Film extends Model implements HasMedia
{
    use HasFactory, InteractsWithMedia;
    public $timestamps = false;

    protected $fillable = [
        "name",
        "synopsis",
        "director",
        "duration",
        "episodes",
        "seasons",
        "type",
        "required_subscription_level",
    ];

    public function registerMediaCollections(): void
    {
        $this->addMediaCollection('images/films')
            ->useFallbackUrl('/images/placeholder.jpg')
            ->useFallbackPath(public_path('/images/placeholder.jpg'));

        $this->addMediaCollection('images2/films')
            ->useFallbackUrl('/images/placeholder.jpg')
            ->useFallbackPath(public_path('/images/placeholder.jpg'));

        $this->addMediaCollection('videos/films');
    }

    public function registerMediaConversions(Media $media = null): void
    {
        if (env('RESIZE_IMAGE') === true) {

            $this->addMediaConversion('resized-image')
                ->width(env('IMAGE_WIDTH', 300))
                ->height(env('IMAGE_HEIGHT', 300));
        }
    }

    public function favoritedBy()
    {
        return $this->belongsToMany(User::class, 'favorites', 'film_id', 'user_id');
    }

    public function categories()
    {
        return $this->belongsToMany(Categoría::class, 'category_film', 'film_id', 'category_id');
    }

    public function suscripciones()
    {
        return $this->belongsToMany(Suscripción::class, 'suscripcion_film', 'film_id', 'suscripcion_id');
    }

    public function users()
    {
        return $this->belongsToMany(User::class, 'user_film', 'film_id', 'user_id');
    }
}

```

En el caso de este modelo, la tabla films, utilizamos el fillable y tal y como se observa, establecemos distintas relaciones N:M con algunas de las tablas de la base de datos. Aún así, esto se explicará con detalle más adelante.

6. 2. 3. Controladores:

Otra parte fundamental a la hora de desarrollar el back-end de nuestra página web, son los controladores, aquellos archivos PHP que se encargan principalmente de manejar las solicitudes HTTP entrantes y que definen cómo se deben responder.

Los controladores son los intermediarios entre las rutas definidas en la aplicación y la lógica que manejan estas. Estos siguen el sistema de MVC (modelo-vista-controlador), donde el propio controlador se encarga de manejar toda la lógica de la aplicación y coordinar la interacción entre el modelo y la vista

COMANDO: php artisan make:Controller nombre_del_controlador

Otro aspecto muy importante a destacar sobre Laravel, es el hecho de que utiliza la sintaxis de Eloquent (ORM), cuyas funciones o métodos nos permiten interactuar directamente y de forma rápida y sencilla con las bases de datos (all, show, findOrFail, store, save...).

A continuación, se muestran las funciones que componen el controlador para obtener los datos de la tabla films o relacionados con esta:

```
<?php

namespace App\Http\Controllers\api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Film;
use App\Models\Categoría;
use App\Models\Suscripción;
use App\Models\UserFilm;
use App\Models\User;
use Carbon\Carbon;

class FilmsController extends Controller
{
    // Listar películas
    public function index(){
        $films = Film::with('media')->get();

        return $films;
    }

    public function getOnlyFilmsContent(){
        $filmsOnly = Film::with('media')->where('type', 'film')->get();

        return $filmsOnly;
    }

    public function getOnlySeriesContent(){
        $filmsOnly = Film::with('media')->where('type', 'serie')->get();

        return $filmsOnly;
    }

    // Listar series
    public function getSeries(){
        $series = Film::with('media')
            ->where('type', '=', 'serie')
            ->get();

        return $series;
    }

    public function getFilmByCat($id){
        $filmByCat = Film::with('media')->where('categoría_id', $id)->get();
        return $filmByCat;
    }
}
```

Desarrollo de aplicaciones web

```

public function getFilmsByCats($id)
{
    if ($id > 0) {
        $query = Film::select('films.*')
            ->join('category_film', 'films.id', '=', 'category_film.film_id')
            ->where('category_film.category_id', $id)
            ->where('films.type', 'film') // Additional where condition for type
            ->with('media');

        $films = $query->get();

        return $films;
    } else {
        $films = Film::with('media')->where('type', 'film')->get();
        return $films;
    }
}

public function getSeriesByCats($id)
{
    if ($id > 0) {
        $query = Film::select('films.*')
            ->join('category_film', 'films.id', '=', 'category_film.film_id')
            ->where('category_film.category_id', $id)
            ->where('films.type', 'serie') // Additional where condition for type
            ->with('media');

        $films = $query->get();

        return $films;
    } else {
        $films = Film::with('media')->where('type', 'serie')->get();
        return $films;
    }
}

```

```

public function store(Request $request){
    $request->validate([
        'name' => 'required',
        'synopsis' => 'required',
        'director' => 'required',
        'duration' => 'required',
        'episodes' => 'required',
        'seasons' => 'required',
        'type' => 'required',
        'suscripcion_id' => 'array',
        'categoria_id' => 'array', // Asegúrate de validar que el campo es un array
    ]);

    // Crea la película
    $filmData = $request->except(['categoria_id', 'thumbnail', 'thumbnail2', 'video']);
    $filmData['punctuation'] = 0;
    $speli = Film::create($filmData);

    // Asigna las categorías
    $categorias = $request->input('categoria_id', []);
    $speli->categories()->attach($categorias);

    /*$categorias = $request->input('categoria_id', []);
    foreach ($categorias as $categoriaId) {
        $speli->categories()->attach($categoriaId);
   }*/

    $suscripcion = $request->input('suscripcion_id', []);
    $speli->suscripciones()->attach($suscripcion);

    // Maneja la carga de medios
    if ($request->hasfile('thumbnail')) {
        $speli->addMediaFromRequest('thumbnail')->preservingOriginal()->toMediaCollection('images-films');
    }

    if ($request->hasFile('thumbnail2')) {
        $speli->addMediaFromRequest('thumbnail2')->preservingOriginal()->toMediaCollection('images2-films');
    }

    if ($request->hasFile('video')) {
        $speli->addMediaFromRequest('video')
            ->preservingOriginal()
            ->toMediaCollection('videos-films', 'video');
    }

    return response()->json(['success' => true, 'data' => $speli]);
}

```

```

// Eliminar pelicula
public function destroy($id){
    $filmDelete = Film::find($id);
    $filmDelete->delete();
    return response()->json(['success' => true, 'data' => 'Serie eliminada correctamente']);
}

// Actualizar pelicula
public function update($id, Request $request){
    $film = Film::find($id);

    $request->validate([
        'name' => 'required',
        'synopsis' => 'required',
        'director' => 'required',
        'punctuation' => 'required',
        'duration' => 'required',
        'video' => 'video',
        'poster' => 'poster'
    ]);

    $dataToUpdate = $request->all();
    $film->update($dataToUpdate);

    return response()->json(['success' => true, 'data' => $film]);
}

public function getFilmById($id) {
    $film = Film::with('media')->find($id);

    if (!$film) {
        return response()->json(['error' => 'Película no encontrada'], 404);
    }

    return $film;
}

public static function getSeriesNameById($id)
{
    $serie = Film::find($id);

    if (!$serie) {
        return response()->json(['error' => 'Película no encontrada'], 404);
    }

    return response()->json(['name' => $serie->name]);
}

public function getAllSeriesNames()
{
    $series = Film::where('type', 'serie')->orderBy('name')->select('id', 'name')->get();
    return $series;
}

```

```

public function getFilmsIds()
{
    $movieIds = Film::pluck('id')->toArray();
    return $movieIds;
}

public function getNews(){
    // Obtener los últimos 5 registros de películas con sus medios asociados
    $news = Film::with('media')->orderBy('id', 'desc')->take(5)->get();

    return $news;
}

public function getRandomFilm()
{
    $randomFilm = Film::with('media')->inRandomOrder()->first();
    return $randomFilm;
}

public function getLastInsertedFilmId() {
    $lastInsertedFilm = Film::orderBy('id', 'desc')->first(); // Ordena por id de forma descendente
    return $lastInsertedFilm->id; // Retorna el ID del Último registro insertado
}

public function addFilmCategoryRelation($filmId, $categoryId)
{
    // Verificar si existen la película y la categoría
    $film = Film::find($filmId);
    $category = Categoria::find($categoryId);

    if (!$film || !$category) {
        return response()->json(['error' => 'Película o categoría no encontrada'], 404);
    }

    // Añadir la relación en la tabla pivot
    $film->categories()->attach($category);

    return response()->json(['success' => 'Relación película-categoría añadida exitosamente']);
}

public function addFilmSuscripcionRelation($filmId, $suscripcionId)
{
    // Verificar si existen la película y la categoría
    $film = Film::find($filmId);
    $suscripcion = Suscripcion::find($suscripcionId);

    if (!$film || !$suscripcion) {
        return response()->json(['error' => 'Película o categoría no encontrada'], 404);
    }

    // Añadir la relación en la tabla pivot
    $film->suscripciones()->attach($suscripcion);

    return response()->json(['success' => 'Relación película-categoría añadida exitosamente']);
}

```

```

// Buscar pelis en el buscador
public function getFilmContentByContentName($name) {
    $contents = Film::where('films.name', 'like', '%' . $name . '%')
        ->where('type', 'film')
        ->with('media')
        ->get();

    if ($contents->isEmpty()) {
        return [];
    }
    return $contents;
}

// Buscar series en el buscado
public function getSerieContentByContentName($name) {
    $contents = Film::where('films.name', 'like', '%' . $name . '%')
        ->where('type', 'serie')
        ->with('media')
        ->get();

    if ($contents->isEmpty()) {
        return [];
    }
    return $contents;
}

public function getFavoriteContentByContentName($name) {
    $contents = Film::where('films.name', 'like', '%' . $name . '%')
        ->with('media')
        ->get();

    if ($contents->isEmpty()) {
        return [];
    }
    return $contents;
}

public function getContentByContentCategoryId($id) {
    $contents = Film::where('films.id', 'like', '%' . $name . '%')
        ->with('media') // Incluye la relación media
        ->get();

    if ($contents->isEmpty()) {
        return [];
    }
    return $contents;
}

public function getContByContId($c_id)
{
    $cont = Film::with('media')
        ->with('categories')
        ->where('id', $c_id)
        ->first();

    return response()->json($cont);
}

```

```
public function deleteViewedFilm($id){
    $userId = auth()->user()->id;
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['error' => 'Usuario no encontrado'], 404);
    }

    // Utiliza el método detach para eliminar la película vista de la tabla pivot
    $user->films()->detach($id);

    return response()->json(['success' => true, 'message' => 'Película vista eliminada con éxito']);
}

public function storeViewedFilm($film_id)
{
    $user_id = auth()->user()->id;

    // Obtener la película
    $film = Film::find($film_id);

    if (!$film) {
        return response()->json(['error' => 'Película no encontrada'], 404);
    }

    // Verificar si el usuario ya ha visto esta película
    if ($film->users()->where('user_id', $user_id)->exists()) {
        return response()->json(['message' => 'El usuario ya ha visto esta película'], 200);
    }

    try {
        // Añadir el registro usando attach
        $film->users()->attach($user_id, ['viewed_date' => Carbon::now()->toDateString()]);

        return response()->json(['message' => 'Película vista registrada exitosamente'], 200);
    } catch (\Exception $e) {
        return response()->json(['error' => 'Error al registrar la película vista'], 500);
    }
}
```

6. 2. 4. Rutas:

Tras crear los modelos y los controladores, toca establecer las rutas que se encargarán de definir o asociar una URL específica a una acción, dentro de nuestra aplicación web. En otras palabras, la ruta indica qué función ejecutar cuando una solicitud HTTP llega a una URL en particular.

En laravel, las rutas las añadiremos en el archivo api.php y de la siguiente manera:

```
Route::post('forget-password', [ForgotPasswordController::class, 'sendResetLinkEmail'])->name('forget.password.post');
Route::post('reset-password', [ResetPasswordController::class, 'reset'])->name('password.reset');

Route::get('tasks', [TaskController::class, 'index']);
Route::post('tasks/', [TaskController::class, 'store']);
Route::put('tasks/update/{id}', [TaskController::class, 'update']);
Route::delete('tasks/{id}', [TaskController::class, 'destroy']);

// Acciones admin películas
Route::get('films', [FilmsController::class, 'index']);

Route::post('films/', [FilmsController::class, 'store']);
Route::put('films/update/{id}', [FilmsController::class, 'update']);
Route::delete('films/{id}', [FilmsController::class, 'destroy']);
Route::get('filmsIds', [FilmsController::class, 'getFilmsIds']);
Route::get('news', [FilmsController::class, 'getNews']);

Route::get('films-only', [FilmsController::class, 'getOnlyFilmsContent']);
Route::get('series-only', [FilmsController::class, 'getOnlySeriesContent']);
Route::get('favorites-only/{user_id}', [FavoritesController::class, 'getOnlyFavoriteContent']);

// Funciones buscador películas, series, favoritos
Route::get('films/{name}', [FilmsController::class, 'getFilmContentByContentName']);
Route::get('series/{name}', [FilmsController::class, 'getSerieContentByContentName']);
Route::get('favorites/{name}', [FilmsController::class, 'getFavoriteContentByContentName']);

Route::get('random-film', [FilmsController::class, 'getRandomFilm']);
Route::get('lastFilm', [FilmsController::class, 'getLastInsertedFilmId']);
Route::get('getSeries', [FilmsController::class, 'getSeries']);

// Añadir películas en la tabla de user_film
Route::post('store/{user_id}/{film_id}', [UserFilmController::class, 'storeViewedFilm']);
Route::get('viewed-films/{user_id}', [UserFilmController::class, 'getUsersViewedFilms']);

// Acciones admin películas
Route::get('episodes', [EpisodesController::class, 'index']);
Route::post('episodes/', [EpisodesController::class, 'store']);
Route::put('episodes/update/{id}', [EpisodesController::class, 'update']);
Route::delete('episodes/{id}', [EpisodesController::class, 'destroy']);
Route::get('episodesBySeason/{id}', [EpisodesController::class, 'getEpisodesBySeasonId']);

// Mostrar película por ID
Route::get('filmsPlayer/{id}', [FilmsController::class, 'getFilmById']);
Route::get('films/{idCat}', [FilmsController::class, 'getFilmByCat']);

Route::get('films/category/{idCategoria}', [FilmsController::class, 'getFilmsByCats']);
Route::get('series/category/{idCategoria}', [FilmsController::class, 'getSeriesByCats']);
Route::get('favorites/category/{idCategoria}/{user_id}', [FavoritesController::class, 'getFavoritesByCats']);

Route::get('getContentById/{c_id}', [FilmsController::class, 'getContById']);
```

De esta forma, cada vez que realicemos una petición en nuestra web, ésta seguirá las rutas que establezcamos y nos permitirá ejecutar las funciones necesarias para su buen funcionamiento.

Recordar que cada ruta requiere un método (GET, POST, PUT, DELETE...), el cual variará según el tipo de función que se quiera ejecutar:

1. GET: Para solicitar datos al servidor.
2. POST: Para enviar datos al servidor.
3. PUT: Para actualizar un recurso existente en el servidor.
4. DELETE: Para eliminar un recurso específico en el servidor.

6. 2. 4. 1. Peticiones al servidor

Como se ha mencionado en el punto anterior, para obtener los datos almacenados en nuestra base de datos, se requiere el uso de peticiones, las cuales realizaremos mediante axios, una biblioteca de JavaScript que nos permitirá gestionar promesas y respuestas asíncronas y manipular la información de una forma más dinámica e interactiva.

En la mayoría de los casos, para el uso de axios, necesitaremos declarar variables reactivas, las cuales actualizarán su valor en función de la petición que realicemos:

```
const isInViewingSection = ref(true);
const isLoading = ref(true);
const route = useRouter();

const films = ref([]);
const randomFilm = ref({media:[]});
const news = ref([]);
const film = ref({media:[]});
const swal = inject('$swal');
const infoModal = ref(null);
const { categoryList, getCategoryList } = useCategories();
const store = useStore();
const user = computed(() => store.state.auth.user)
const filmPunct = ref([]);

const favorites = ref([]);
const favoritesById = ref([]);
const allViewedUsersFilms = ref([]);

const selectedFilm = ref(null); // Variable para almacenar la película seleccionada
const seasonOptions = ref([]); // Lista de opciones para el menú desplegable

const seasons = ref([]);
const filteredSeasons = ref([]);
const filteredContent = ref([]);
const active = ref(0);
const episodesBySeason = ref([]);
const episodes = ref([]);
const review = ref({});
const reviews = ref([]);
const InfoSubsFilm = ref([]);
```

Dicho valor, se aplicará según la respuesta que se obtenga mediante la URL que se le pase al axios:

```
axios.get('/api/films/')
  .then(response => {
    isLoading.value = false;
    | films.value = response.data;
  })
  .catch(error => {
    isLoading.value = false;
    console.error('Error fetching films:', error);
  });

```

En este caso, la URL que se le pasa al axios, está vinculada con una de las rutas que se ha mencionado anteriormente, la cual llama a la función index de FilmsController:

```
Route::get('films', [FilmsController::class, 'index']);
```

6. 2. 5. Relaciones N:M

Con el objetivo de mantener una buena infraestructura, sólida y sin redundancia de datos, decidimos añadir ciertas relaciones N:M, las cuales requieren de ciertos elementos de eloquent para su correcto funcionamiento.

En nuestro caso, disponemos de tres relaciones de muchos a muchos, USER_FILM, CATEGORY_FILM y SUBSCRIPTION_FILM, las cuales disponen de sus respectivas claves primarias, además de ciertos campos adicionales no primarios, como en el caso de CATEGORY_FILM con el creation_date o USER_FILM con el viewed_date.

Como se ha mencionado anteriormente, para establecer este tipo de relaciones, se han usado ciertos elementos o sintaxis de eloquent como el belongsToMany, pero sobretodo, el attach, el sync o el detach, los cuales nos permiten manejar las relaciones entre dos modelos y las tablas asociadas.

En este caso tenemos el attach, el cual lo utilizamos en la función que se encarga de almacenar las películas en la base de datos. Tal y como se observa en la imagen, mediante el attach podemos asignar o asociar las películas que creamos con las categorías existentes en la base de datos.

De esta forma, nos aseguramos que una película puede tener muchas categorías y que una categoría puede estar asociada a muchas películas. Lo mismo sucede con el tema de las suscripciones.

```
public function store(Request $request){
    $request->validate([
        'name' => 'required',
        'synopsis' => 'required',
        'director' => 'required',
        'duration' => 'required',
        'episodes' => 'required',
        'seasons' => 'required',
        'type' => 'required',
        'suscripcion_id' => 'array',
        'categoria_id' => 'array', // Asegúrate de validar que el campo es un array
    ]);

    // Crea la película
    $filmData = $request->except(['categoria_id', 'thumbnail', 'thumbnail2', 'video']);
    $filmData['punctuation'] = 0;
    $peli = Film::create($filmData);

    // Asigna las categorías
    $creationDate = now()->toDateString();
    $categorias = $request->input('categoria_id', []);
    $peli->categories()->attach($categorias, ['creation_date' => $creationDate]);

    $suscripcion = $request->input('suscripcion_id', []);
    $peli->suscripcions()->attach($suscripcion);

    // Maneja la carga de medios
    if ($request->hasFile('thumbnail')) {
        $peli->addMediaFromRequest('thumbnail')->preservingOriginal()->toMediaCollection('images-films');
    }

    if ($request->hasFile('thumbnail2')) {
        $peli->addMediaFromRequest('thumbnail2')->preservingOriginal()->toMediaCollection('images2-films');
    }

    if ($request->hasFile('video')) {
        $peli->addMediaFromRequest('video')
            ->preservingOriginal()
            ->toMediaCollection('videos-films', 'video');
    }

    return response()->json(['success' => true, 'data' => $peli]);
}
```

Ahora tenemos el sync, que nos permitirá sincronizar los modelos asociados con los parámetros que reciba la función:

```
public function subscribe(Request $request, $subscriptionId)
{
    $user = Auth::user();
    $days= $request->input("days");

    // Sincronizar la suscripción para el usuario
    $user->suscripciones()->sync([$subscriptionId]);

    // Obtener la suscripción asociada al usuario
    $subscription = $user->suscripciones()->where('suscripcion_id', $subscriptionId)->first();

    // Obtener la fecha actual
    $today = Carbon::today();

    // Calcular la fecha de finalización basada en la duración
    $endDate = $today->copy()->addDays($days);

    // Actualizar las fechas de inicio y finalización de la suscripción
    $subscription->pivot->start_date = $today;
    $subscription->pivot->end_date = $endDate;
    $subscription->pivot->save();
}
```

Por último, el detach, con el cual, podremos eliminar todas aquellas relaciones entre modelos en una relación de muchos a muchos:. En este caso, lo utilizamos para eliminar las películas de la tabla pivot entre usuarios y películas.

```
public function deleteViewedFilm($id){
    $userId = auth()->user()->id;
    $user = User::find($userId);

    if (!$user) {
        return response()->json(['error' => 'Usuario no encontrado'], 404);
    }

    // Utiliza el método detach para eliminar la película vista de la tabla pivot
    $user->films()->detach($id);

    return response()->json(['success' => true, 'message' => 'Película vista eliminada con éxito']);
}
```

6. 2. 6. Funcionalidades extra

6. 2. 6. 1. Firebase - Realtime Database

Además de la base de datos MySQL proporcionada por XAMPP, con el objetivo de indagar más en profundidad y aportar herramientas adicionales a nuestra plataforma, decidimos llevar a cabo la creación de un sistema de gestión de listas de reproducción.

Para ello, utilizamos Firebase, una plataforma digital de Google que facilita el desarrollo de aplicaciones, de forma fácil y eficiente. No obstante, el principal motivo por el que nos decidimos por este método fue para poder crear una base de datos en tiempo real y aprender a gestionar los datos de esta.

El funcionamiento es muy sencillo:

Empezaremos accediendo a la página web oficial de Firebase, para poder crear la cuenta con la que gestionaremos todos los proyectos y posteriormente, deberemos acceder al apartado de Realtime Database y crear nuestro proyecto, donde se almacenarán todos los datos.

Tras la configuración del nuevo proyecto, entraremos en la sección de configuración y copiaremos los datos que nos proporciona, los cuales los tendremos que añadir en nuestro proyecto para poder establecer la conexión entre este y la base de datos:

```
import { initializeApp } from "firebase/app";
import { getAnalytics } from "firebase/analytics";
import { getDatabase, ref, onValue, set } from "firebase/database";

// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
// For Firebase JS SDK v7.20.0 and later, measurementId is optional

const firebaseConfig = {
  apiKey: "AIzaSyDtP660c3nbdzWtrOwX301eGQ5Dh_YKMqE",
  authDomain: "iwatch-e996c.firebaseio.com",
  projectId: "iwatch-e996c",
  storageBucket: "iwatch-e996c.appspot.com",
  messagingSenderId: "588749122433",
  appId: "1:588749122433:web:f029f81a2103087dcc9680",
  measurementId: "G-QQ6EEMLK7",
  databaseURL: https://iwatch-e996c-default-rtdb.europe-west1.firebaseio.app/
};

const db = initializeApp(firebaseConfig);

const datab = getDatabase()

export { datab };
```

Una vez tengamos la configuración correctamente, podremos empezar a programar las funciones principales para el CRUD de la base de datos en tiempo real.

```
// Leer datos de la bbdd
const listener = ref(db, 'playlists/')
onValue(listener, (snapshot) => {
  const data = snapshot.val();
  allData.value = data;
});

const userPlaylists = ref([]);

const listener2 = ref(db, 'mis-playlists')
onValue(listener2, (snapshot) => {
  const data = snapshot.val();
  userPlaylists.value = data ? Object.values(data) : [];
})

const filteredUserPlaylists = computed(() => {
  return userPlaylists.value.filter(playlist => playlist.user_id === user.value.id);
});
```

```
// Crear nueva playlist
const addPlaylist = (array) => {
  if(array.length > 0){
    console.log("LARGO BIEN");
    const newPlaylist = {
      playlist_name: playlistName.value,
      movies: array,
      user_id: 1,
    };
    set(ref(db, `playlists/${newPlaylist.playlist_name}`), newPlaylist);
    swal({
      allowOutsideClick: true,
      timer: 2000,
      timerProgressBar: true,
      customClass: {
        timerProgressBar: 'succes-progress-bar'
      },
      position: "top-end",
      icon: 'success',
      title: 'Playlist creada!',
      showConfirmButton: false
    });
  }else{
    swal({
      allowOutsideClick: true,
      timer: 2000,
      timerProgressBar: true,
      customClass: {
        timerProgressBar: 'error-progress-bar'
      },
      position: "top-end",
      icon: 'error',
      title: 'Añade la información necesaria!',
      showConfirmButton: false
    });
  }
};
```

```
//Funcion para crear mi playlist utilizando el set
const addMyPlaylist = (array, playlistName) => {
  if(array.length > 0){
    console.log("LARGO BIEN");
    const newPlaylist = {
      playlist_name: playlistName,
      movies: array,
      user_id: user.value.id,
    };
    set(reff(db, `mis-playlists/${newPlaylist.playlist_name}`), newPlaylist);
    swal({
      allowOutsideClick: true,
      timer: 2000,
      timerProgressBar: true,
      customClass: {
        timerProgressBar: 'succes-progress-bar'
      },
      position: "top-end",
      icon: 'success',
      title: 'Playlist añadida a favoritos!',
      showConfirmButton: false
    });
  }else{
    swal({
      allowOutsideClick: true,
      timer: 2000,
      timerProgressBar: true,
      customClass: {
        timerProgressBar: 'error-progress-bar'
      },
      position: "top-end",
      icon: 'error',
      title: 'No se ha podido añadir',
      showConfirmButton: false
    });
  }
};
```

```
//Funcion para eliminar la playlist
const deletePlaylist = (playlist) => {
  const playlistRef = reff(db, `mis-playlists/${playlist.playlist_name}`);
  remove(playlistRef)
  .then(() => {
    swal({
      allowOutsideClick: true,
      timer: 2000,
      timerProgressBar: true,
      customClass: {
        timerProgressBar: 'succes-progress-bar'
      },
      position: "top-end",
      icon: 'success',
      title: 'Playlist eliminada de Mis playlist!',
      showConfirmButton: false
    });
  })
  .catch(error) => {
    console.error(`Error al eliminar la playlist "${playlist.playlist_name}":`, error);
    swal({
      allowOutsideClick: true,
      timer: 2000,
      timerProgressBar: true,
      customClass: {
        timerProgressBar: 'succes-progress-bar'
      },
      position: "top-end",
      icon: 'error',
      title: 'Error!',
      showConfirmButton: false
    });
  });
};
```

Mencionar que en este apartado hemos añadido las funciones más importantes para gestionar las interacciones entre el usuario y la base de datos en tiempo real, el resto de funciones tales como llamar acciones mediante botones, por ejemplo, estarán presentes en el código.

6. 2. 6. 2. Tareas de programación

En nuestra plataforma, los usuarios necesitan una suscripción para poder disfrutar del contenido. No obstante, dichas suscripciones tienen una fecha de expiración, es decir, caducan al cabo de cierto tiempo (uno, dos o tres meses).

Para ello, hemos hecho uso de las tareas de programación o tareas cron, las cuales se programan para ejecutarse en intervalos específicos de tiempo, como por ejemplo cada minuto, hora, diariamente, mensualmente, etc.

Estas se establecen mediante la herramienta Artisan y el método ‘schedule()’, en el archivo Kernel.php, en el directorio app\Console.

```
<?php

namespace App\Console;

use Illuminate\Support\Facades\Log;
use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

use Carbon\Carbon;

use App\Models\UserSubscription;

class Kernel extends ConsoleKernel
{
    /**
     * Define the application's command schedule.
     *
     * @param \Illuminate\Console\Scheduling\Schedule $schedule
     * @return void
     */
    protected function schedule(Schedule $schedule)
    {
        $schedule->call(function(){
            $today = Carbon::today('Europe/Madrid');
            // Obtener registros cuya fecha de finalización coincide con la fecha de hoy
            $subscriptions = UserSubscription::whereDate('end_date', $today)->get();
            // Log de la cantidad de registros encontrados
            Log::info('Se encontraron ' . $subscriptions->count() . ' registros con fecha de finalización de hoy.');
            // Eliminar registros
            $deleted = UserSubscription::whereDate('end_date', $today)->delete();
            // Log de la cantidad de registros eliminados
            Log::info('Se eliminaron ' . $deleted . ' registros con fecha de finalización de hoy.');
        })->daily();
    }

    /**
     * Register the commands for the application.
     *
     * @return void
     */
    protected function commands()
    {
        $this->load(__DIR__.'/Commands');

        require base_path('routes/console.php');
    }
}
```

Tal y como se observa en la imagen, declaramos la función schedule dentro del kernel, la cual ejecutará lo siguiente:

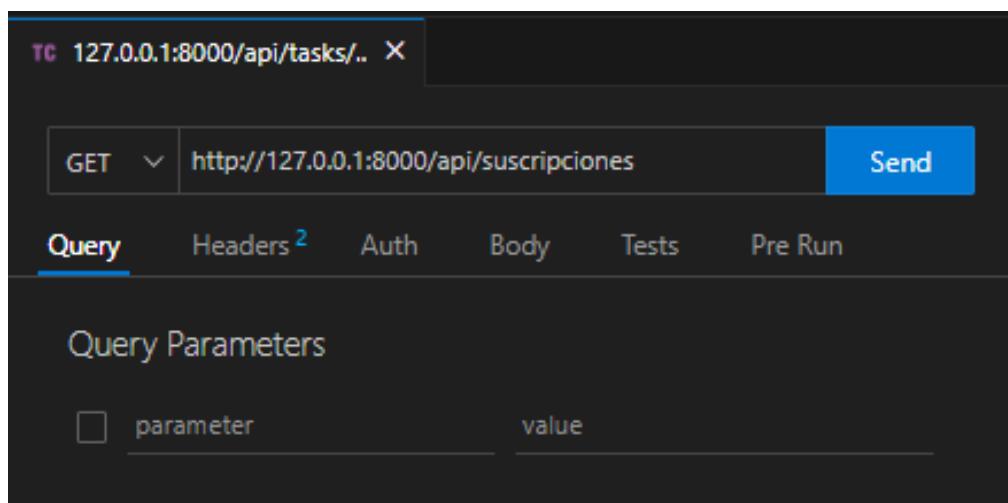
1. Se declara el carbon, para obtener la fecha de hoy en la zona horaria de Europa/Madrid.
2. Luego, se obtienen todos los registros cuya fecha de finalización coincide con la de hoy y posteriormente se eliminan, asegurándonos de que el plazo de la suscripción del usuario ha expirado correctamente.

Mencionar que, en este caso, dicha funcionalidad se ejecutara diariamente.

6. 2. 7. Thunder client

Para optimizar el tiempo y mejorar el uso de las funciones, decidimos utilizar la extensión Thunder Client, un gestor de peticiones que dispone de una interfaz simple y muy visual, lo cual facilita la comprensión de los datos.

El funcionamiento es muy sencillo: Creamos una nueva request, determinamos el método (GET, POST, PUT, DELETE...) y añadimos la ruta que manda la petición. Posteriormente enviamos la petición y esperamos a los resultados en formato JSON.



Response	Headers 7	Cookies	Results	Docs
1 [2 { 3 "id": 1, 4 "name": "Básico", 5 "price": "9.95", 6 "duration": "17:07:13", 7 "resolution": "SD", 8 "quality": "Baja" 9 }, 10 { 11 "id": 2, 12 "name": "Estándar", 13 "price": "14.95", 14 "duration": "17:07:13", 15 "resolution": "HD", 16 "quality": "Media" 17 }, 18 { 19 "id": 3, 20 "name": "Premium", 21 "price": "19.95", 22 "duration": "17:07:13", 23 "resolution": "4K", 24 "quality": "Alta" 25 } 26]				

Gracias a esta extensión, podemos verificar la información con la que trabajan nuestros controladores.

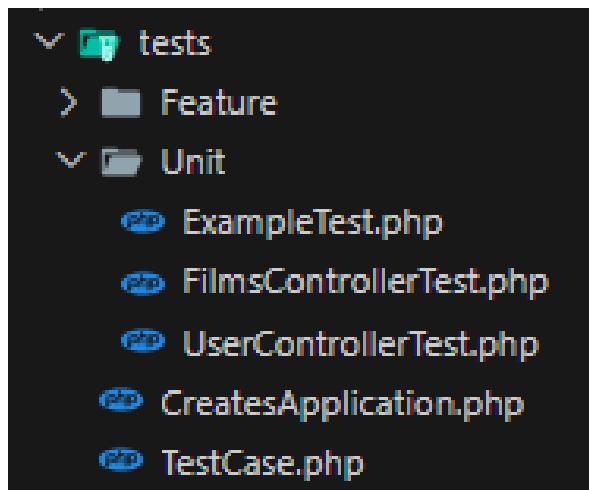
6. 2. 8. Pruebas automatizadas

Con el objetivo de verificar el comportamiento de nuestro código de forma individual y asegurarnos del correcto funcionamiento de la plataforma, hemos realizado ciertas pruebas automatizadas con las que poner a prueba las clases de la aplicación, los modelos y los controladores entre otros.

Para empezar, necesitamos crear el archivo necesario en el que almacenar las funciones que ejecutarán los tests individuales mediante el siguiente comando:

```
php artisan make:test nombre_del_test
```

Tras ejecutar dicho comando, se nos creará el archivo automáticamente y en el directorio `tests/Unit`.



A continuación, se muestran las pruebas unitarias que hemos realizado:

Pruebas del controlador FilmsController.php

```
<?php

namespace Tests\Unit;

use Illuminate\Foundation\Testing\RefreshDatabase;
use Illuminate\Foundation\Testing\DatabaseTransactions;
use Tests\TestCase;
use App\Http\Controllers\Api\FilmsController;
use App\Models\Film;

class FilmsControllerTest extends TestCase
{
    use DatabaseTransactions;

    /**
     * A basic unit test example.
     */
    public function testIndex()
    {
        // Obtén algunas películas existentes en la base de datos
        $existingFilms = Film::all();

        if ($existingFilms->isEmpty()) {
            $this->markTestSkipped('No hay películas en la base de datos para probar');
        }

        // Crea una instancia del controlador FilmsController
        $controller = new FilmsController();

        // Llama al método index del controlador
        $result = $controller->index();

        // Verifica que el resultado no sea nulo
        $this->assertNotNull($result);

        // Verifica que el resultado sea una instancia de Illuminate\Support\Collection
        $this->assertInstanceOf(\Illuminate\Support\Collection::class, $result);

        // Verifica que el número de películas devueltas es igual al número de películas existentes
        $this->assertCount($existingFilms->count(), $result);
    }
}
```

```
public function testGetFilmByIdExistingFilm()
{
    // Obtener una película existente de la base de datos
    $existingFilm = Film::inRandomOrder()->first();

    // Asegurarse de que haya al menos una película en la base de datos
    if (!$existingFilm) {
        $this->markTestSkipped('No hay películas en la base de datos para probar');
    }

    // Crear una instancia del controlador FilmsController
    $controller = new FilmsController();

    // Llamar a la función getFilmById con el ID de la película seleccionada
    $result = $controller->getFilmById($existingFilm->id);

    // Verificar que el resultado no sea nulo
    $this->assertNotNull($result);

    // Verificar que el resultado sea la película seleccionada
    $this->assertEquals($existingFilm->id, $result->id);
}
```

```
public function testGetFilmByIdNonExistingFilm()
{
    // ID de una película que no existe
    $nonExistingFilmId = 9999;

    // Crear una instancia del controlador FilmsController
    $controller = new FilmsController();

    // Llamar a la función getFilmById con el ID de la película no existente
    $result = $controller->getFilmById($nonExistingFilmId);

    // Verificar que el resultado sea un mensaje de error y un código de estado 404
    $this->assertEquals(404, $result->status());
    $this->assertEquals('Película no encontrada', $result->getData()->error);
}
```

Pruebas del controlador UserController.php

```
<?php

namespace Tests\Unit;

use Tests\TestCase;

use App\Models\User;
use App\Models\Role;
use Illuminate\Http\UploadedFile;
use Illuminate\Support\Facades\Storage;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UserControllerTest extends TestCase
{
    /**
     * A basic unit test example.
     */

    use RefreshDatabase;

    public function test_example(): void
    {
        $this->assertTrue(true);
    }

    public function it_updates_user_profile()
    {
        $user = User::factory()->create();

        $this->actingAs($user);

        $newUserData = [
            'name' => 'John',
            'email' => 'john@example.com',
            'apellido' => 'Doe',
            'phone' => '123456789'
        ];

        $response = $this->put(route('user.update'), $newUserData);

        $this->assertDatabaseHas('users', [
            'id' => $user->id,
            'name' => $newUserData['name'],
            'email' => $newUserData['email'],
            'apellido' => $newUserData['apellido'],
            'phone' => $newUserData['phone']
        ]);

        $response->assertStatus(200);

        $response->assertJson(['message' => 'Perfil actualizado correctamente']);
    }
}
```

```

public function it_stores_a_new_user()
{
    // Simulamos un usuario autenticado
    $this->actingAs(User::factory()->create());

    // Datos de prueba para crear un nuevo usuario
    $userData = [
        'name' => 'John',
        'apellido' => 'Doe',
        'email' => 'john@example.com',
        'password' => 'password123',
        // 'role_id' => Role::factory()->create()->id,
        'phone' => '123456789',
        // Agrega más campos si es necesario para representar tus datos de entrada
    ];

    // Simulamos la carga de un archivo de imagen de perfil
    Storage::fake('public');
    $file = UploadedFile::fake()->image('profile.jpg');

    // Hacemos una solicitud al método 'store' del controlador UserController
    $response = $this->post(route('user.store'), $userData + ['profile_image' => $file]);

    // Verificamos que el usuario se haya almacenado correctamente en la base de datos
    $this->assertDatabaseHas('users', [
        'name' => $userData['name'],
        'apellido' => $userData['apellido'],
        'email' => $userData['email'],
        'phone' => $userData['phone'],
        // Agrega más verificaciones según la estructura de tu base de datos
    ]);

    // Verificamos que el usuario tenga asignado el rol adecuado
    $this->assertDatabaseHas('role_user', [
        'user_id' => User::where('email', $userData['email'])->first()->id,
        'role_id' => $userData['role_id'],
    ]);

    // Verificamos que se haya almacenado la imagen de perfil correctamente
    Storage::disk('public')->assertExists('images2/profile/' . $file->hashName());

    // Verificamos que la respuesta sea exitosa
    $response->assertStatus(200);

    // Verificamos que la respuesta contenga los datos del usuario creado
    $response->assertJson([
        'name' => $userData['name'],
        'apellido' => $userData['apellido'],
        'email' => $userData['email'],
        'phone' => $userData['phone'],
        // Agrega más campos si es necesario según la respuesta esperada
    ]);
}
}

```

7. INFORMACIÓN ADICIONAL

Durante el desarrollo de este proyecto, han surgido diversos problemas en cuanto al back-end, los cuales nos ha parecido correcto mencionar en este manual y con el objetivo de mostrar las formas o posibles soluciones que hemos encontrado:

7. 1. Error 429 (Too many requests)

Debido al gran flujo de datos y a un volumen excesivo de peticiones realizadas al servidor en un período determinado de tiempo, a veces nos aparecía el error 429. Como consecuencia, en muchas ocasiones teníamos problemas al cargar la plataforma, a veces se quedaba colgada o colapsaba y había que refrescar.

Tras buscar información, descubrimos que Laravel dispone de una opción para quitar el límite de peticiones al servidor o bien de aumentarla. En este caso, solo tuvimos que acceder al archivo Kernel.php, en el directorio de app\Http y comentar la línea del throttle:api.

```
protected $middlewareGroups = [
    'web' => [
        \App\Http\Middleware\EncryptCookies::class,
        \Illuminate\Cookie\Middleware\AddQueuedCookiesToResponse::class,
        \Illuminate\Session\Middleware\StartSession::class,
        \Illuminate\View\Middleware\ShareErrorsFromSession::class,
        \App\Http\Middleware\VerifyCsrfToken::class,
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
    ],
    'api' => [
        \Laravel\Sanctum\Http\Middleware\EnsureFrontendRequestsAreStateful::class,
        // 'throttle:api',
        \Illuminate\Routing\Middleware\SubstituteBindings::class,
        \Rakutentech\LaravelRequestDocs\LaravelRequestDocsMiddleware::class,
    ],
];
```

Tras este cambio en el código, solucionamos el problema con dicho error.

7. 1. Gestión de archivos multimedia

iWatch es una plataforma online cuyo producto principal son elementos multimedia, ya sean imágenes o vídeos.

Para poder hacer uso de estos elementos mediante Laravel, utilizamos Spatie Media Library, una biblioteca que facilita el manejo y almacenamiento de dichos archivos en aplicaciones.

Respecto a su funcionamiento, tuvimos que añadir ciertas funcionalidades al código con tal de relacionar el contenido de la base de datos junto con sus respectivos archivos multimedia, además de ejecutar determinados comandos en la terminal como por ejemplo:

php artisan storage:link

Dicho comando, se encarga de crear un enlace simbólico entre la carpeta de almacenamiento storage y la carpeta public/storage, permitiendo acceder a los archivos almacenados en la carpeta de almacenamiento a través de la carpeta pública, facilitando de esta forma, la entrega de archivos estáticos a los usuarios a través de la web.

Por otro lado, tenemos las funciones añadidas en el código:

```
public function registerMediaCollections(): void
{
    $this->addMediaCollection('images/films')
        ->useFallbackUrl('/images/placeholder.jpg')
        ->useFallbackPath(public_path('/images/placeholder.jpg'));

    $this->addMediaCollection('images2/films')
        ->useFallbackUrl('/images/placeholder.jpg')
        ->useFallbackPath(public_path('/images/placeholder.jpg'));

    $this->addMediaCollection('videos/films');
}

public function registerMediaConversions(Media $media = null): void
{
    if (env('RESIZE_IMAGE') === true) {
        $this->addMediaConversion('resized-image')
            ->width(env('IMAGE_WIDTH', 300))
            ->height(env('IMAGE_HEIGHT', 300));
    }
}
```

En este caso, la primera función se encarga de crear las colecciones de archivos junto con su URL asociada, mientras que la segunda redimensiona los archivos que se añaden.

Este es el sistema que hemos seguido para poder añadir tres tipos de colecciones al proyecto: El póster vertical, el horizontal y el video de cada película/serie que se reproduzca.

8. RESULTADO FINAL

Tras tres meses de proyecto, se han alcanzado todos los objetivos que se marcaron al principio de este, además de todos los requisitos iniciales, indispensables para el desarrollo de la plataforma.

Tal y como se especificó en los objetivos iniciales, el objetivo principal era el de crear una plataforma cinematográfica, con un funcionamiento similar al de otras plataformas similares como Netflix, Disney+ o HBO.

En este caso, el resultado final ha sido bastante gratificante, ya que, se ha logrado el desarrollo de una plataforma con la que poder visualizar contenido audiovisual, además de otras funcionalidades como por ejemplo la de creación de reseñas (para mantener cierta interacción para con el contenido de la web), creación de playlist públicas (para compartir los gustos de los usuarios), un sistema de suscripciones, etcétera.

En general, se ha logrado una plataforma funcional que cumple con los requisitos planteados al principio del proyecto.

9. CONCLUSIONES

El inicio de este proyecto fue algo difuso, estábamos un poco perdidos y no teníamos mucha idea sobre cómo enfocar este proyecto, debido a la complejidad de

este y al uso de las nuevas tecnologías de desarrollo web empleadas para ello, como por ejemplo Laravel o Vue.

Como en los trabajos anteriores, empezamos con una toma de decisiones donde tuvimos que determinar la temática o el enfoque de nuestra plataforma, en cuyo caso, acabó siendo sobre el mundo cinematográfico.

Tras determinar el tipo de aplicación que queríamos desarrollar, empezamos la fase del diseño o los esbozos iniciales, para hacernos una idea básica e inicial sobre cómo sería la estructura de la web y de las secciones que la conformarían. Determinamos las vistas y algunos de los elementos gráficos de la página, como por ejemplo los menús, los carruseles o los paneles de información.

Posteriormente y con el objetivo de empezar a establecer las bases del proyecto, nos pusimos con el desarrollo de la base de datos, determinando los tipos de datos de esta y la estructura que seguiría, mediante el desarrollo de diagramas.

Por último y tras tener una idea inicial sobre las bases de nuestra plataforma web, nos pusimos a desarrollar el código, tanto la parte del front-end como del back-end, sin duda la parte más complicada y larga.

A lo largo de estos meses, hemos tenido muchas complicaciones a la hora de crear una plataforma adecuada, teniendo en cuenta la complejidad de esta, sobre todo a la hora de almacenar o usar tantos recursos como por ejemplo los videos o las imágenes de alta resolución. Además, en cuanto al diseño de la web, hemos intentado aplicar un diseño limpio, moderno y de fácil uso para cualquier usuario, basándonos y usando otras plataformas similares a modo de inspiración.

Como conclusión final, creemos que tras todo el esfuerzo y tiempo empleado, se ha llegado a los objetivos esperados y marcados desde el principio. Y en cuanto a las experiencias personales, nos ha gustado mucho trabajar en dicho proyecto, ya que nos ha permitido aprender muchos aspectos tanto de Laravel, Vue o bien otras tecnologías como FireBase entre otras.

10. WEBGRAFIA

LARAVEL: <https://laravel.com/docs/11.x>

STACK OVERFLOW: <https://stackoverflow.com/>

ERROR 429 (Too Many Requests): [Solución a error 429: Acelerando tus peticiones en Laravel con Throttle](#)  ([linkedin.com](#))

LARACASTS: <https://laracasts.com/discuss>