

IES FRANCISCO DE GOYA - LA ELIPA



# FilmXtra

---

**PROYECTO FIN DEL CICLO FORMATIVO DESARROLLO DE  
APLICACIONES WEB**

REALIZADO POR:

VÍCTOR FERNÁNDEZ GORMAZ

ELENA GATA JIMÉNEZ

RODRIGO SUÁREZ PARTIDA



07/06/2023

# ÍNDICE

<b>1. Introducción</b>	<b>4</b>
1.1. Objetivos iniciales	4
1.2. Tecnologías	4
1.2.1. Laravel	4
1.2.2. Tailwind CSS	5
1.2.3. Flowbite	5
1.2.4. MySQL y MySQL Workbench	5
1.2.5. JetBrains Toolbox	6
1.2.6. PhpStorm	6
1.2.7. Vue.js	7
1.2.8. Visual Studio Code	7
1.2.9. GitHub	8
1.2.10. Herramientas de desarrollo de Chrome	8
1.2.11. HTML5 y CSS3	9
1.2.12. Bootstrap	9
1.2.13. ChatGPT	10
1.2.14. Google OAuth	10
1.2.15. Servidor SMTP de Gmail	10
1.2.16. Inertia JS	11
<b>2. Análisis de requisitos</b>	<b>11</b>
2.1. Prototipos e ideas iniciales	11
<b>3. Desarrollo del diseño web</b>	<b>16</b>
3.1. Página "Welcome"	18
3.2. Página "Top"	22
3.3. Página "Obra"	25
3.4. Página "Top Valoraciones"	28
3.5. Página "Ficha Valoraciones"	29
3.6. Página "Login" y "Register"	32
3.7. Página "Edit"	34
3.8. Página "Error 404"	35
3.9. Páginas de verificación y restauración de contraseñas	36
<b>4. Adecuación del entorno de trabajo</b>	<b>38</b>
4.1. Instalación y configuración del IDE	38
4.2. Andamiaje del proyecto	40
4.3. Control de versiones	42
<b>5. Arquitectura del proyecto</b>	<b>44</b>
5.1. Modelos	44
5.2. Migraciones	47
5.3. Breeze(autenticación)	48
5.4. Rutas	50
5.5. Controladores	51
5.6. Vistas, Flujo de información y JavaScript	53

5.7. Paginación	54
5.8. Middleware	56
5.9. Validación	56
5.10. Ignition y la depuración	58
<b>6. Desarrollo del proyecto</b>	<b>59</b>
6.1. Verificación en dos pasos	59
6.2. Google OAuth 2	73
<b>7. Desarrollo de la base de datos</b>	<b>78</b>
7.1. Adopción y diseño de la base de datos para la persistencia	78
7.2. Inserts. Información almacenada	80
<b>8. Conclusión</b>	<b>81</b>
<b>9. Proyectos a futuro</b>	<b>82</b>
9.1. Podcast	82
9.2. Foto de usuario	82
9.3. Listas	82
9.4. Honeypot	82
9.5. Mejora del botón de búsqueda	82
9.6. Valoraciones y críticas de usuarios	82
9.7. Nuevos mensajes interactivos	83
9.8. Añadir fotos de las películas	83
9.9. Segundo tráiler	83
9.10. Roles	83
9.11. Notificaciones	83
9.12. Laravel Policies	83
9.13. Información sobre cines	84
9.14. Organización del código	84
9.15. Despliegue	84
9.16. Ordenamiento	84
9.17. Sesiones	84
9.18. Tienda	84
9.19. Inicio de sesión con GitHub	84
9.20. Introducir más películas	84
<b>10. Tareas FilmXtra</b>	<b>85</b>
<b>11. Bibliografía</b>	<b>86</b>
11.1. Documentación para la memoria	86
11.2. Documentación del back-end (Víctor)	87
11.3. Documentación del diseño web (Elena)	87
11.4. Documentación de redes (Rodrigo)	88

# 1. Introducción

Este documento constituye la memoria del Proyecto de Fin de Ciclo Formativo de Desarrollo de Aplicaciones Web (DAW), llevado a cabo por Víctor Fernández Gormaz, Elena Gata Jiménez y Rodrigo Suárez Partida.

Nosotros, como equipo de FilmXtra, hemos trabajado con el objetivo de diseñar y desarrollar una página web interactiva enfocada en el fascinante mundo del cine.

## 1.1. Objetivos iniciales

Nuestro principal propósito es construir una página web desde cero utilizando nuevas tecnologías de programación como Laravel, Tailwind o Google OAuth 2, y usando el cine como temática principal.

Para lograrlo, establecimos los siguientes objetivos iniciales:

- Proporcionar información detallada sobre diferentes películas, estableciendo el núcleo temático de la web.
- Permitir la participación de los usuarios mediante la calificación de películas, creando así un sistema de colaboración entre usuarios.
- Habilitar la opción para que los usuarios puedan realizar críticas de películas, fomentando una interacción más elaborada.
- Permitir que los usuarios creen listas de películas favoritas basadas en género, época o año específico.
- Implementar la posibilidad de comprar productos relacionados con el cine, con el objetivo de monetizar la web, además de la publicidad.
- Ofrecer la opción de compra de películas como una posible forma adicional de monetización.
- Permitir que los usuarios se registren y mantengan sus sesiones activas en la plataforma.

Con estos objetivos iniciales, nos comprometimos a desarrollar una página web completa y atractiva, utilizando tecnologías modernas y brindando a los usuarios una experiencia interactiva y enriquecedora en el mundo del cine.

## 1.2. Tecnologías

En el desarrollo de nuestra página web, hemos utilizado una combinación de lenguajes de programación, tecnologías y entornos para crear una plataforma robusta y funcional. Algunos de los principales utilizados son:

### 1.2.1. Laravel

 Laravel es un framework de código abierto para desarrollar aplicaciones y servicios web con PHP. Su filosofía es desarrollar código PHP de forma elegante y simple, evitando el “código espagueti”. Fue creado en 2011 y tiene una gran influencia de frameworks como Ruby on Rails, Sinatra y ASP.NET MVC.

En nuestro caso, utilizamos Laravel como el motor principal del backend de nuestra aplicación. Laravel es un framework de PHP que desempeña un papel fundamental en la gestión de los datos de nuestra aplicación. Actúa como una capa intermedia entre la capa de persistencia de datos y las vistas en Vue.js, nuestro frontend. Una de las herramientas clave que utilizamos para lograr esta integración entre Laravel y Vue.js es Inertia.

Laravel se encarga de obtener los datos necesarios de la capa de persistencia y procesarlos de acuerdo con nuestras necesidades. Luego, estos datos son enviados a las vistas de Vue.js a través de Inertia, permitiéndonos construir interfaces interactivas y dinámicas para nuestros usuarios.

En resumen, Laravel actúa como el motor que impulsa nuestro backend, procesando y transformando los datos, mientras que Inertia nos permite integrar de manera eficiente Laravel y Vue.js para crear una experiencia fluida y consistente en toda nuestra aplicación.

### 1.2.2. Tailwind CSS

 Tailwind CSS es una framework de CSS de código abierto para el diseño de páginas web. La principal característica de esta biblioteca es que, a diferencia de otras como Bootstrap, no genera una serie de clases predefinidas para elementos como botones o tablas.

En el proyecto FilmXtra, hemos incorporado esta tecnología de forma integral en el diseño de todas las páginas de nuestro sitio web. Hemos tenido en cuenta cada detalle, desde los posters y títulos hasta los textos y botones, para asegurarnos de que todo esté cuidadosamente diseñado y en armonía. Esta tecnología ha sido fundamental para crear una experiencia visualmente atractiva y coherente en nuestro sitio web.

### 1.2.3. Flowbite

 Flowbite es una biblioteca de código abierto de componentes web de Tailwind CSS construida con clases. También incluye elementos interactivos como componentes, páginas, etcétera.

Esta página ha sido una fuente de inspiración incalculable para nuestro proyecto, brindándonos ideas que hemos adaptado de manera personalizada. Hemos incorporado componentes como la barra de navegación y el pie de página de esta página, pero los hemos ajustado a nuestros propios estilos, colores y preferencias para garantizar una integración perfecta con nuestra visión única. De esta manera, hemos logrado fusionar la inspiración obtenida con nuestra identidad y estética, creando una experiencia visual coherente y distintiva en nuestra página web.

### 1.2.4. MySQL y MySQL Workbench

 MySQL es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: Licencia pública general/Licencia comercial por Oracle Corporation y está considerada como la base de datos de código abierto más popular del mundo, y una de las más populares en general junto a Oracle y Microsoft SQL Server.

En cambio, MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos y gestión y mantenimiento para el sistema de base de datos de MySQL.

MySQL es el sistema de gestión de bases de datos que hemos utilizado en nuestro proyecto para almacenar todos los datos de manera persistente. Es el encargado de gestionar y organizar la información de nuestras bases de datos.

Por otro lado, MySQL Workbench es una herramienta que hemos utilizado de manera opcional para facilitar la visualización y diseño de nuestras tablas de base de datos. Es una interfaz gráfica que nos permite crear y modificar esquemas de bases de datos, así como generar diagramas visuales de las relaciones entre tablas.

Es importante destacar que MySQL Workbench no es una herramienta necesaria para el desarrollo de un proyecto, ya que el sistema de gestión de bases de datos MySQL puede ser utilizado sin ella. Sin embargo, su utilización puede ser beneficiosa para tener una representación más visual y esquemática de nuestras tablas y relaciones en la base de datos, lo que facilita el diseño y mantenimiento de la estructura de datos.

### 1.2.5. JetBrains Toolbox



Toolbox de JetBrains se refiere a la lista de herramientas de escritorio disponibles de JetBrains para desarrolladores, con una suscripción mensual o anual. Actualmente, están disponibles los siguientes productos con la suscripción a Toolbox de JetBrains: AppCode, CLion.

JetBrains ha sido una plataforma fundamental para la integración de PhpStorm en nuestro trabajo. Gracias a esta aplicación, hemos logrado una colaboración eficiente como equipo, ya que cuenta con una herramienta que permite la integración directa con GitHub. Esto nos ha brindado la posibilidad de trabajar utilizando el sistema de control de versiones de Git, lo que ha facilitado enormemente nuestra capacidad para trabajar de manera simultánea sin enfrentar dificultades.

La integración de GitHub en PhpStorm ha sido especialmente valiosa, ya que nos ha permitido mantener nuestros archivos y proyectos organizados y actualizados en todo momento. Con esta funcionalidad, hemos podido sincronizar nuestros cambios, colaborar en tiempo real y resolver conflictos de manera más fluida. Además, la capacidad de trabajar simultáneamente en diferentes ramas y fusionar nuestros cambios de forma segura ha mejorado significativamente nuestra eficiencia y productividad como equipo.

En resumen, gracias a JetBrains y su integración con GitHub, hemos podido aprovechar al máximo PhpStorm y trabajar de manera colaborativa y eficiente como equipo. Esta herramienta ha sido clave para optimizar nuestro flujo de trabajo y garantizar una gestión efectiva del control de versiones en nuestros proyectos.

### 1.2.6. PhpStorm



PhpStorm es reconocido por su Depurador visual libre de configuración, lo que le permite obtener un conocimiento extraordinario sobre lo que sucede dentro de su aplicación a cada paso. Funciona con Xdebug y Zend Debugger, y se puede utilizar tanto local como remotamente.

Incorporamos PhpStorm a nuestro flujo de trabajo a través de JetBrains Toolbox, como mencionamos anteriormente. Aprovechamos la cuenta gratuita de un año disponible para estudiantes, lo que nos permitió utilizar esta aplicación sin incurrir en costos adicionales, ya que normalmente es una herramienta de pago. Además de ser económicamente ventajoso, hemos podido disfrutar de las diversas funcionalidades y beneficios que PhpStorm ofrece.

Una de las características destacadas de PhpStorm es su depurador visual de alta calidad, el cual ha sido muy beneficioso para nuestro trabajo colaborativo. Esta funcionalidad nos ha permitido identificar y solucionar rápidamente errores y problemas en nuestro código, facilitando la depuración y mejorando la calidad general de nuestro proyecto.

Gracias a la cuenta gratuita para estudiantes y las funcionalidades avanzadas de PhpStorm, hemos podido maximizar nuestra productividad y eficiencia en el desarrollo de aplicaciones web. Esta herramienta ha sido un componente integral en nuestro flujo de trabajo, brindándonos una plataforma sólida y confiable para crear y colaborar en nuestro proyecto de manera efectiva.

### 1.2.7. Vue.js



Vue.js es un framework de JavaScript de código abierto para la construcción de interfaces de usuario y aplicaciones de una sola página. Fue creado por Evan You, y es mantenido por él y por el resto de los miembros activos del equipo central que provienen de diversas empresas como Netlify y Netguru.

En nuestro proyecto web, inicialmente optamos por un enfoque de diseño tradicional utilizando HTML5 y CSS3. Sin embargo, pronto nos dimos cuenta de la importancia de trabajar de manera conjunta y organizada, manteniendo a todos los miembros del equipo actualizados en todo momento. Con este objetivo en mente, decidimos hacer un cambio en nuestra metodología de diseño.

Para lograr una mayor colaboración y eficiencia, tomamos la decisión de migrar a una aplicación que nos permitiera trabajar de manera más integrada. En lugar de seguir con el enfoque tradicional, nos trasladamos a utilizar Vue.js y Tailwind como nuestras herramientas principales.

Este cambio nos ha permitido aprovechar las ventajas de Vue.js, un framework progresivo para la construcción de interfaces de usuario, que nos brinda mayor flexibilidad y modularidad en el desarrollo de nuestra página web. Además, hemos adoptado Tailwind, una biblioteca de estilos utilitarios, que nos proporciona componentes predefinidos y una sintaxis optimizada para el desarrollo rápido y eficiente.

Al realizar esta transición, hemos logrado una mejor coordinación y fluidez en nuestro proceso de diseño, permitiendo que todos los miembros del equipo trabajen de manera conjunta y contribuyan al proyecto de manera más efectiva.

### 1.2.8. Visual Studio Code



Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código.

Al principio, como no estábamos familiarizados con PhpStorm teníamos cierta resistencia a usar esta nueva aplicación, y todo el equipo comenzó a utilizar Visual Studio Code para

escribir los primeros códigos del proyecto. Este editor de código es ampliamente conocido por nosotros, ya que lo hemos utilizado durante todo el año en clase. Sin embargo, una vez nos adentramos en el uso de PhpStorm, dejamos de lado Visual Studio Code. PhpStorm nos brindó funcionalidades que la aplicación de Microsoft no tiene, como la integración con el control de versiones de GitHub, que hemos utilizado ampliamente para trabajar en equipo de manera simultánea.

### 1.2.9. GitHub

 GitHub es una web para alojar proyectos utilizando el sistema de control de versiones de Git, y se utiliza principalmente para la creación de código fuente de programas de ordenador. El software que opera GitHub fue escrito en Ruby on Rails, y desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc.

Para nuestro proyecto, resultó fundamental crear un repositorio en GitHub, dado que éramos un grupo de tres integrantes. Sin tener un lugar centralizado para alojar nuestros archivos, nos habría resultado extremadamente difícil mantenerlos actualizados en todo momento. Con esta necesidad en mente, el día diecisésis de abril, Víctor creó el repositorio "[filmxtra](#)". Posteriormente, creamos tres ramas principales: "main", "consolidada" y "development".

En la rama "main" se encontraba la versión principal del proyecto, donde se desarrollaba la funcionalidad principal. La rama "consolidada" era utilizada por Víctor y Elena, quienes trabajaban de manera más directa en la consolidación y optimización del código. En la rama "development", Rodrigo implementó todo su código relacionado con la verificación en dos pasos a través de correo electrónico, así como las funcionalidades de inicio de sesión utilizando cuentas de Google.

La creación de este repositorio en GitHub y la utilización de ramas específicas nos permitió trabajar de forma colaborativa y organizada, manteniendo un seguimiento claro de los cambios realizados por cada miembro del equipo.

### 1.2.10. Herramientas de desarrollo de Chrome

 La herramienta de desarrollo de Chrome, también conocido como DevTools es un set de herramientas útiles para desarrollo web y herramientas para debug dentro de Chrome. Incluyen paneles para elementos, conectividad, fuentes, línea de tiempo, perfiles, recursos, auditorías y consolas.

Las herramientas que Chrome nos brinda resultan sumamente útiles, tal y como aprendimos y utilizamos en clase. Nos permiten identificar y corregir errores en el código de forma rápida y sencilla, sin poner en riesgo la integridad del proyecto. Además, como mencionamos anteriormente, estas herramientas nos brindan la posibilidad de realizar pruebas desde el panel de elementos y observar los resultados de manera instantánea.

Esta funcionalidad resulta especialmente beneficiosa tanto en el diseño de la página como en aspectos relacionados con redes y el desarrollo del backend. Nos ha permitido optimizar y agilizar nuestro trabajo, ya que podemos realizar ajustes y mejoras de forma eficiente, ahorrando tiempo y esfuerzo.

### 1.2.11. HTML5 y CSS3



HyperText Markup Language versión 5, o más conocido como HTML5. Es la quinta revisión del lenguaje HTML y permite definir los nuevos estándares de desarrollo web, modificando el código existente para solucionar problemas y actualizándose a las nuevas necesidades de hoy en día.

En cambio, Cascading Style Sheets 3 (CSS3), o también conocido como hojas de estilo en cascada, es un lenguaje que maneja el diseño y presentación de las páginas webs, es decir, cómo lucen cuando un usuario las visita. Funciona junto con el lenguaje HTML que se encarga del contenido básico de las páginas.

Inicialmente, decidimos trabajar en el apartado de diseño utilizando Visual Studio Code, haciendo uso de HTML5, CSS3 y Bootstrap. Sin embargo, pronto nos dimos cuenta de que sería más eficiente y beneficioso para el equipo trabajar de manera conjunta en PhpStorm. Elena comenzó a diseñar la página de inicio utilizando HTML5, CSS3 y Bootstrap en Visual Studio Code. Sin embargo, al trasladar su trabajo a PhpStorm, se percató de que era necesario realizar modificaciones en su código. Esto se debió a que PhpStorm utiliza Vue.js como tecnología principal, y Bootstrap no es compatible con ella. Por lo tanto, Elena tuvo que aprender a trabajar con Tailwind en poco tiempo y rediseñar su código para adaptarlo a esta nueva herramienta.

Pero este cambio de enfoque en las tecnologías utilizadas nos permitió aprovechar al máximo las funcionalidades ofrecidas por PhpStorm y mejorar nuestra colaboración como equipo. Aunque supuso un desafío para Elena tener que adaptarse rápidamente a Tailwind, nos permitió contar con una base de código más coherente y optimizada para el desarrollo de la página web.

### 1.2.12. Bootstrap



Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios web y aplicaciones web.

Como mencionamos anteriormente, inicialmente optamos por utilizar Bootstrap en el área de diseño web. Sin embargo, debido al cambio de aplicación y a nuestras necesidades específicas, decidimos dejar de lado Bootstrap y nos inclinamos por trabajar con Tailwind, junto con su biblioteca gratuita y multiplataforma, Flowbite.

La transición a Tailwind y Flowbite fue motivada por varias razones. Estas herramientas nos brindaron una mayor flexibilidad y nos permitieron adaptar el diseño de nuestra página web de manera más precisa a nuestras necesidades específicas. Además, la biblioteca multiplataforma de Flowbite nos proporcionó componentes listos para usar, lo que agilizó significativamente nuestro proceso de diseño y desarrollo.

Este cambio nos permitió tener un mayor control sobre la apariencia y la funcionalidad de nuestra página web, y nos brindó una base sólida para crear una experiencia de usuario atractiva y coherente en todas las plataformas.

En resumen, tras dejar de lado Bootstrap, optamos por trabajar con Tailwind y su biblioteca gratuita Flowbite. Estas herramientas nos brindaron la flexibilidad y las opciones necesarias para adaptar el diseño de nuestra página web a nuestras necesidades específicas.

### 1.2.13. ChatGPT



ChatGPT es un prototipo de chatbot de inteligencia artificial desarrollado en 2022 por OpenAI que se especializa en el diálogo. El chatbot es un gran modelo de lenguaje, ajustado con técnicas de aprendizaje tanto supervisadas como de refuerzo. Se basa en el modelo GPT-4 de OpenAI, una versión mejorada de GPT-3.

Esta aplicación, tan destacada en la actualidad, ha resultado ser extremadamente útil para nuestro equipo a la hora de redactar valoraciones de películas de manera eficiente. Desde el inicio del desarrollo, Víctor se encargó de redactar algunas valoraciones, tanto de profesionales como de usuarios. Luego, cuando Elena finalizó sus tareas de diseño web, se dedicó a completar estas valoraciones. Sin embargo, llegó un punto en el que se quedó sin ideas y decidió recurrir a ChatGPT para que escribiera algunas valoraciones personales en su lugar. Esta aplicación nos brindó la posibilidad de solicitar nuevas valoraciones o ajustar la longitud de los textos en poco tiempo.

En definitiva, esta aplicación ha sido una gran ayuda en términos de ahorro de tiempo y ha contribuido a que las valoraciones de los usuarios sean más objetivas y detalladas. Nos ha permitido agilizar el proceso de redacción, obtener nuevos enfoques y mejorar la calidad de las valoraciones.

### 1.2.14. Google OAuth



OAuth 2 es un framework de autorización que permite a las aplicaciones obtener acceso limitado a las cuentas de usuario de servicios como Google, GitHub, Facebook, Twitter, entre otros. Su principal objetivo es delegar la autenticación de usuarios al servicio que gestiona esas cuentas, lo que permite que las aplicaciones de terceros obtengan acceso controlado a los datos del usuario.

Este framework es versátil y se puede utilizar en diferentes tipos de aplicaciones, como aplicaciones web, móviles e incluso programas de escritorio. En nuestro caso, hemos aprovechado la funcionalidad de OAuth 2 para permitir a los usuarios autenticarse utilizando su cuenta de Google. Esto elimina la necesidad de un registro convencional, lo cual resulta conveniente y rápido para los usuarios. Además, al implementar esta opción de autenticación, podemos lograr atraer a nuevos clientes de manera eficiente.

### 1.2.15. Servidor SMTP de Gmail



El servidor SMTP de Gmail es de uso público y permite enviar correos electrónicos utilizando una cuenta de Gmail y los servidores de Google. Para configurar el envío de correos, hemos utilizado la dirección del servidor SMTP de Gmail, que es '[smtp.gmail.com](mailto:smtp.gmail.com)'. Este servidor admite conexiones TLS en el puerto 587 y SSL en el puerto 465. Recomendamos utilizar TLS, ya que es más seguro y ofrece una mayor confianza en el envío y recepción de correos electrónicos.

En nuestro caso, hemos utilizado la siguiente configuración:

- Servidor SMTP de Gmail: '[smtp.gmail.com](mailto:smtp.gmail.com)'.
- Dirección de correo electrónico utilizada: [filmxtra.23@gmail.com](mailto:filmxtra.23@gmail.com).

- Contraseña: en lugar de utilizar la contraseña principal de la cuenta de Gmail, hemos generado una contraseña de aplicación específica para permitir el envío de correos desde esta cuenta de Gmail.
- Puerto utilizado: hemos optado por el puerto 587 (conexión TLS) para garantizar una mayor seguridad en el envío de correos electrónicos.

Hemos utilizado esta configuración para implementar una verificación en dos pasos durante el proceso de registro de nuevos usuarios, lo que les permite compartir valoraciones y comentarios con otros usuarios de manera segura.

### 1.2.16. Inertia JS

 Inertia JS es una herramienta que ofrece grandes ventajas a los desarrolladores de aplicaciones backend al permitirles crear aplicaciones Single Page Application (SPA) de forma más eficiente. Su enfoque en el renderizado del servidor y la técnica de carga diferida, también conocida como "lazy-loading", brinda transiciones más rápidas y fluidas entre diferentes vistas sin tener que recargar la página completa.

La integración de Inertia JS con frameworks populares como Laravel, en nuestro caso, facilita el intercambio de datos entre el frontend y el backend. Esto significa que podemos utilizar la etiqueta `@Inertia` en nuestros componentes para renderizarlos y comunicarnos con el backend de manera más eficiente.

Al utilizar Inertia JS, podemos aprovechar las capacidades de renderizado en el servidor para obtener una respuesta más rápida y mejorar la experiencia del usuario. Además, el concepto de carga diferida nos permite cargar sólo los componentes necesarios en cada momento, optimizando así el rendimiento de nuestra aplicación.

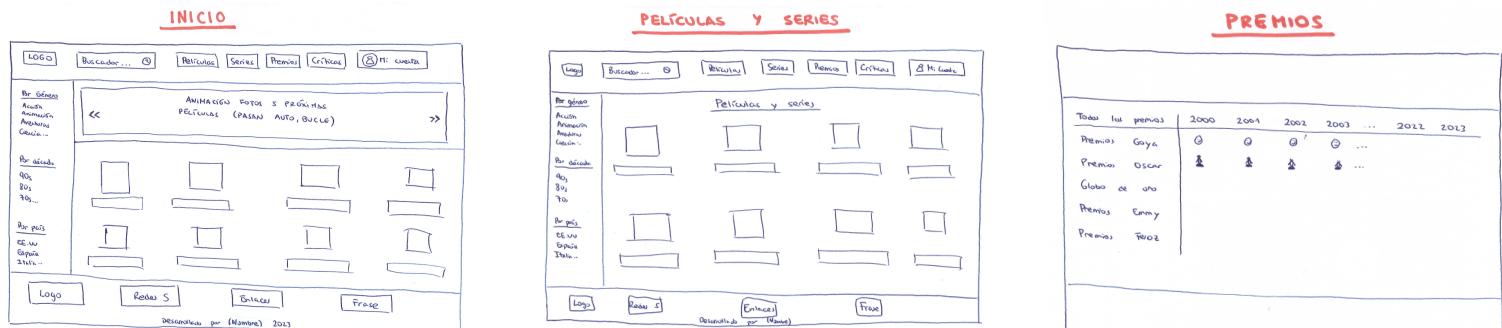
En resumen, Inertia JS nos brinda una forma más eficiente de crear aplicaciones SPA al combinar el renderizado en el servidor y la carga diferida. Su integración con frameworks como Laravel nos permite intercambiar datos de manera efectiva y mejorar la experiencia del usuario.

## 2. Análisis de requisitos

### 2.1. Prototipos e ideas iniciales

En primer lugar, es crucial en cualquier proyecto plasmar las ideas iniciales en papel y compartirlas con el equipo. Esto permite asegurarnos de que las visiones del equipo de diseño web o frontend se alineen con las posibilidades y objetivos del equipo de backend. En el caso de FilmXtra, el departamento de diseño web, representado por Elena, desempeñó un papel fundamental al crear prototipos en papel de las principales páginas de nuestra web.

Elena comenzó por crear tres prototipos en papel que representaban la página de inicio, las secciones de películas y series, y la sección de premios. Estos prototipos se elaboraron con detalle, cómo se puede apreciar en las siguientes imágenes.

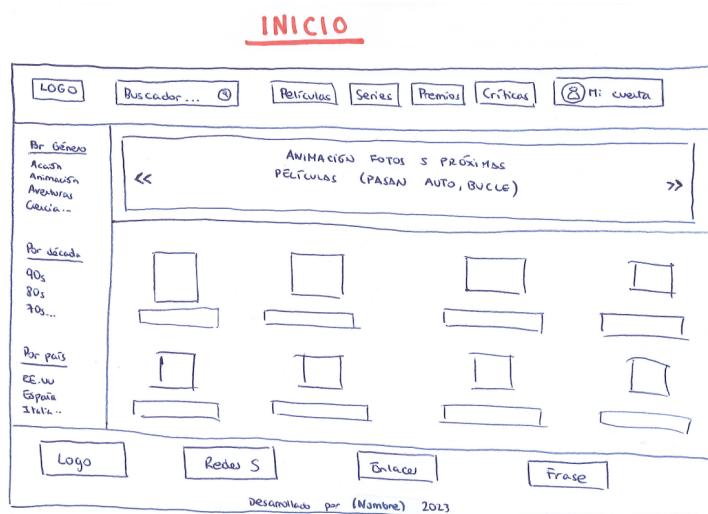


En la página de inicio, teníamos una clara visión de cómo queríamos estructurarla. Implementamos un menú de navegación con el logo distintivo de FilmXtra, diferentes secciones para acceder a diversas páginas, un buscador y un área de cuenta donde los usuarios podrían registrarse, iniciar sesión, cerrar sesión y acceder a su cuenta.

En la siguiente sección de la página, decidimos añadir una animación utilizando un carrusel de cinco imágenes para proporcionar dinamismo y captar la atención del usuario. Inicialmente, la idea era mostrar avances de próximas películas, pero tras una revisión de diseño, optamos por cambiar el enfoque y asegurarnos de que todo el aspecto visual de la página mantuviera la misma sintonía y coherencia.

Justo debajo del carrusel, se presentaron los posters y títulos de las películas que FilmXtra tenía en su base de datos en filas de cuatro elementos cada una. Además, diseñamos una columna en el lateral para facilitar la interacción con los usuarios, donde se mostraban todos los géneros, épocas y años disponibles de nuestra base de datos.

Por último, incluimos un pie de página, que contenía nuestro logo, enlaces a nuestras redes sociales y otra información relevante sobre FilmXtra. Esta sección tenía como objetivo brindar a los usuarios una forma sencilla de acceder a más información y mantenerse en contacto con nosotros.



En el prototipo inicial de la página de películas y series de nuestro proyecto, teníamos la intención de proporcionar información tanto sobre películas como sobre series. Sin embargo, más adelante, en el desarrollo del proyecto, decidimos enfocarnos únicamente en

películas. Esto se debió a que utilizamos una base de datos existente que había sido diseñada exclusivamente para películas, directores, reparto, secuelas, precuelas, etc. Pero todos estos detalles de la base de datos los abordaremos más adelante.

Siguiendo con el diseño de la página de inicio, decidimos mantener el menú de navegación y el pie de página consistentes en todas las páginas de la web de FilmXtra. Sin embargo, la columna lateral que muestra los datos de género, época y años solo se encontraría en la página de inicio y en la página de películas. En otras páginas, no tendría mucho sentido y no sería relevante.

Como se puede observar, esta pantalla se asemeja mucho a la página de inicio. La única diferencia es que en esta página quisimos dar más importancia a los posters de las películas, para que los usuarios solo accedieran a esta pantalla si deseaban consultar información específica sobre una película en particular.



El último prototipo creado por Elena, fue el dirigido a la pantalla de premios. En un principio, se consideró mostrar información sobre las películas en relación a los premios y festivales en los que habían participado. Sin embargo, más adelante esta idea fue descartada cuando Víctor, desde el área de backend, sugirió presentar la información de cada película de manera más concisa y clara.

La propuesta de Víctor consistía en que el usuario pudiera obtener todos los datos necesarios de manera rápida y fácil con solo un vistazo a la pantalla. Esto implicó que la pantalla de premios sufriría importantes cambios a lo largo del desarrollo del proyecto para adaptarse a esta visión más centrada en proporcionar una visión general de cada película.

PREMIOS	
Todos los premios	2000 2001 2002 2003 ... 2022 2023
Premios Goya	0 0 0 0 ...
Premios Oscar	1 1 1 1 ...
Globo de Oro	
Premios Emmy	
Premios Tony	

En segundo lugar, en el área de diseño web nos propusimos encontrar un nombre adecuado para nuestro proyecto. Ya que, inicialmente, en el anteproyecto se eligió el nombre de "Cinema Paradiso" debido a la base de datos que utilizamos se creó bajo ese nombre, pero sentíamos la necesidad de darle una identidad propia. Con la ayuda de nuestra imaginación y de ChatGPT, pudimos encontrar el mejor nombre para nuestro proyecto: "FilmXtra".



Sin embargo, encontrar el nombre adecuado requiere un proceso. Utilizamos la IA de ChatGPT para obtener cuatro ideas de nombres relacionados con una página web para películas, donde los usuarios pudieran registrarse, valorar y criticar las películas. Las cuatro opciones que obtuvimos fueron: "Películas al Día", "Cinéfilos en Acción", "La Butaca de Cine" y "FilmXtra". Después de una votación en grupo, el nombre ganador fue "FilmXtra".

Posteriormente, realizamos otra votación relacionada con la paleta de colores que queríamos utilizar en la web y el logo. Inicialmente, el azul fue el color elegido, pero Elena, al crear el logo y la web, se decantó por el rojo. Esta elección se basó en la asociación del cine con el color rojo, representado por las butacas, las alfombras rojas de los festivales, los envases de palomitas, entre otros elementos.

Por último, con todas las ideas del equipo recopiladas, nos pusimos manos a la obra. En poco tiempo, Elena presentó al grupo tres propuestas para el logo y el nombre de la web, utilizando la herramienta Canva. Después de una cuidadosa consideración, la tercera opción resultó ser la ganadora, tal como se puede apreciar en la versión final de la web presentada.

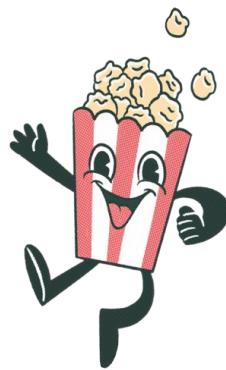


Sin embargo, no nos detuvimos ahí. A muchos miembros del equipo no les convencía por completo la tipografía del logo, por lo que se presentaron seis prototipos finales adicionales.

Después de una cuidadosa deliberación, el quinto prototipo resultó ser el ganador, y a partir de ahí se crearon los logotipos finales.



Una vez seleccionado el logotipo, los colores y la tipografía, Elena desarrolló tres versiones finales del logo para sus diferentes usos. La primera versión muestra el logo completo con la tipografía en color negro, la segunda versión es igual que la anterior pero, pero con la tipografía en color blanco, y la tercera y última versión corresponde al favicon que se coloca en la imagen de la pestaña superior del navegador.



Una vez completadas todas las etapas anteriores, llegamos al último paso crucial: el prototipado digital, y final.

En esta fase, desde el área de diseño web, se trasladaron los prototipos realizados en papel a Figma, una poderosa herramienta basada principalmente en generar prototipos para las páginas web.

En este paso, es de suma importancia comprender lo que se puede y no se puede lograr, ya que refleja en gran medida el aspecto final de cada pantalla. Como se puede apreciar en las siguientes imágenes, se crearon prototipos en Figma para cuatro pantallas principales: la página de inicio, la visualización de todas las películas, la pantalla detallada de una película

específica al hacer clic en ella, y una nueva pantalla para el inicio de sesión y registro de usuarios en nuestro sitio web.

**PÁGINA DE INICIO**

**PÁGINA DE PELÍCULAS**

**PÁGINA DE UNA PELÍCULA EN CONCRETO**

**PÁGINA DE LA CUENTA**

Una vez completada esta etapa y tras discutirlo con todo el equipo de FilmXtra, todos teníamos claro cuáles eran nuestras tareas a realizar. Víctor se encargó de preparar toda la estructura del proyecto en Laravel, aprovechando la licencia gratuita para estudiantes en PhpStorm. Comenzó a crear las ramas y organizar todo el entorno de trabajo. Elena investigó la posibilidad de utilizar Wordpress para implementar el diseño, pero finalmente determinamos que no era viable. Por lo tanto, siguió adelante con la idea original y continuó trabajando en el diseño desde cero. Por su parte, Rodrigo se centró en estudiar las variables de correo electrónico, realizar pruebas y configurar todo lo relacionado con la verificación en dos pasos. Cada miembro del equipo asumió su responsabilidad y comenzó a trabajar en sus respectivas áreas para avanzar en el proyecto de FilmXtra.

### 3. Desarrollo del diseño web

Después de haber revisado y documentado el proceso de creación de cada una de las páginas, vamos a explicar detalladamente cómo hemos desarrollado cada una de ellas y las razones detrás de nuestras decisiones. A lo largo de estos tres meses, Elena ha desplegado su talento en el diseño y la programación para llevar a cabo este proyecto. Por ello, a continuación, desglosaremos el proceso de creación de cada página, que consta de tres etapas: el diseño del prototipo en papel, la creación del prototipo digital en Figma y, finalmente, la implementación de la página final en Vue.js.

En primer lugar, a finales de marzo, se diseñaron prototipos en papel de cada una de las páginas que se planeaban desarrollar. Estos prototipos permitieron visualizar la estructura y el diseño general de cada página, brindando una idea clara de cómo se organizaría la información y cómo interactuaría el usuario con la interfaz.

Luego, a principios de abril, se trasladaron estos prototipos a la plataforma de diseño digital Figma. Esta herramienta permitió crear prototipos interactivos y detallados, en los que se definieron los elementos visuales, los estilos y las interacciones de cada página. Figma facilitó la colaboración entre los miembros del equipo, permitiendo realizar ajustes y mejoras de manera ágil.

Una vez finalizados los prototipos en Figma, Elena comenzó a trabajar en la implementación de las páginas finales utilizando el framework Vue.js. Con este enfoque basado en componentes, se pudo desarrollar de manera eficiente y escalable cada una de las páginas del sitio web. Elena utilizó sus habilidades en la programación frontend para traducir los diseños y las interacciones definidas en Figma en código funcional.

Para mantener una mejor organización y sincronización entre el frontend y el backend, Víctor y Elena utilizaron una hoja de cálculo compartida en Google. Esta herramienta les permitió tener un registro centralizado de las páginas y funcionalidades del proyecto, así como establecer prioridades y alinear las ideas del backend.

En la hoja de cálculo, ambos colaboradores pudieron agregar y describir las diferentes páginas y características que debían ser implementadas. También utilizaron columnas para indicar el estado de desarrollo, las dependencias entre páginas y cualquier información relevante para su correcta implementación.

Esta colaboración en la hoja de cálculo proporcionó una visión clara de las tareas pendientes, evitó confusiones y permitió una comunicación efectiva entre el frontend y el backend. Además, les permitió mantener un seguimiento actualizado del progreso del proyecto y realizar ajustes en función de las necesidades y prioridades establecidas.

PÁGINA	NOMBRE ELENA	RUTA VÍCTOR (uri -> name)	HTML	Comentario	Estado
Inicio/Home	Welcome	uri: '/ -> name: '/'	Layout principal (header, nav y footer), y unas películas generadas al azar	Vista de bienvenida, general	Enviada e incorporada
Película en concreto	Obra	uri: '/obra/[titulo]' -> name: 'obra'	Info de la obra, críticas de usuarios, botones para críticas, evaluar, like de críticas etc	Ficha de una obra, con su info e interactiva en función de usuario logueado	Enviada e incorporada
Género en concreto	Top	uri: '/filtradas/[ criterio ]' -> name: 'filtradas'	Listas de las películas filtradas por género, época, país o filtro personalizado	Muestra las películas según un criterio, con una barra lateral con género, época y país	Enviada e incorporada
Cuenta de usuario	Edit	uri: '/cuenta/[id]' -> name: 'cuenta'	Edición de la información del usuario: nombre, contraseña, etc....	Página de usuario con campos editables	Enviada e incorporada
404 error	Error404		Error 404, frase, fondo y gif	Página de error que se muestra cuando la ruta no es la correcta	Enviada e incorporada
Inicio sesión	Login		Email, contraseña, botones y link para registrarse	Página para iniciar sesión	Enviada e incorporada
Crear cuenta	Register		Email, contraseña, confirmación contraseña, fecha nacimiento, botones y link inicio	Página de registrarse	Enviada e incorporada
Resetear contraseña	ResetPassword		Email, nueva contraseña, confirmación nueva contraseña, botón y link inicio	Página de resetear la contraseña	Enviada e incorporada
Confirmar contraseña	ConfirmPassword		Contraseña, botón y link inicio	Página de confirmación de contraseña	Enviada e incorporada
Contraseña olvidada	ForgotPassword		Email, botón y link inicio	Página restablecer contraseña	Enviada e incorporada
Verificar email	VerifyEmail		Email, botón y link inicio	Página de verificación email para acceder a la cuenta	Enviada e incorporada
Valoraciones obras	TopValoraciones		Valoraciones usuarios y críticos con poster y botón datos obra	Página de valoraciones de obras	Enviada e incorporada
		Enviada e incorporada			
		En curso			
		En mente			

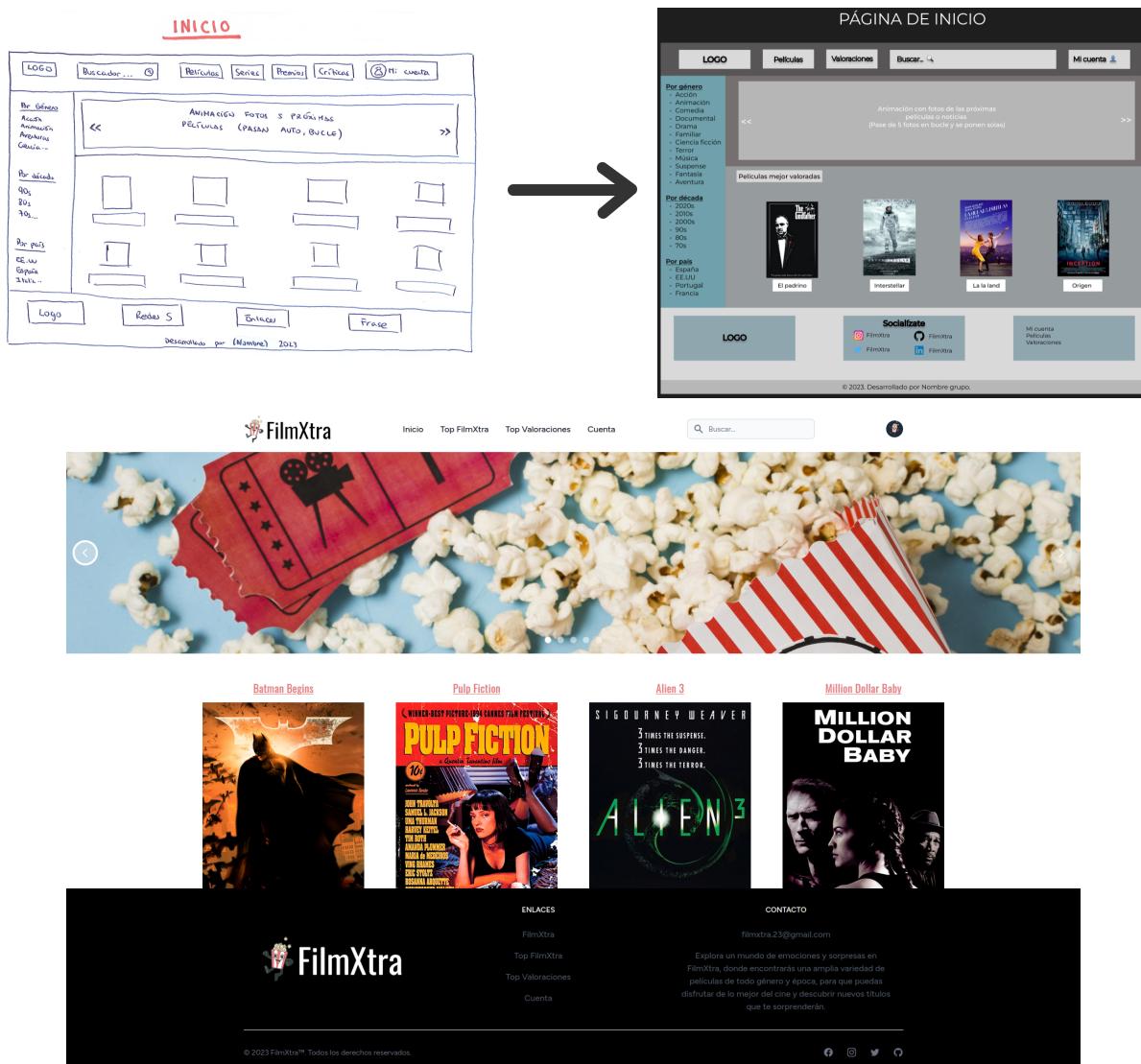
### 3.1. Página “Welcome”

La página de inicio ha sido una de nuestras prioridades desde el principio, y nos complace ver cómo ha evolucionado a lo largo del proceso. Hemos buscado mantener una estructura clara y una apariencia coherente en cada una de las versiones que hemos desarrollado. Nuestro objetivo principal era crear una página de bienvenida que transmitiera alegría, dinamismo, pero que al mismo tiempo fuera seria y minimalista.

Desde las primeras etapas de diseño en papel hasta la versión final implementada, hemos mantenido la esencia y la claridad en la página de inicio, ya que, queríamos asegurarnos de que los usuarios se sintieran atraídos por la presentación visual y encontraran fácilmente la información que buscaban.

Hemos trabajado en armonizar los elementos visuales, los colores y la tipografía para lograr una apariencia agradable y atractiva, por ello, la estructura de la página se ha mantenido consistente, brindando una navegación intuitiva y una experiencia de usuario fluida.

La página de inicio es el punto de entrada a nuestro sitio web, por lo que era fundamental captar la atención de los visitantes y presentarles de manera clara y concisa qué pueden encontrar en FilmXtra. Estamos realmente satisfechos con el resultado final, ya que hemos logrado transmitir la esencia que buscábamos.

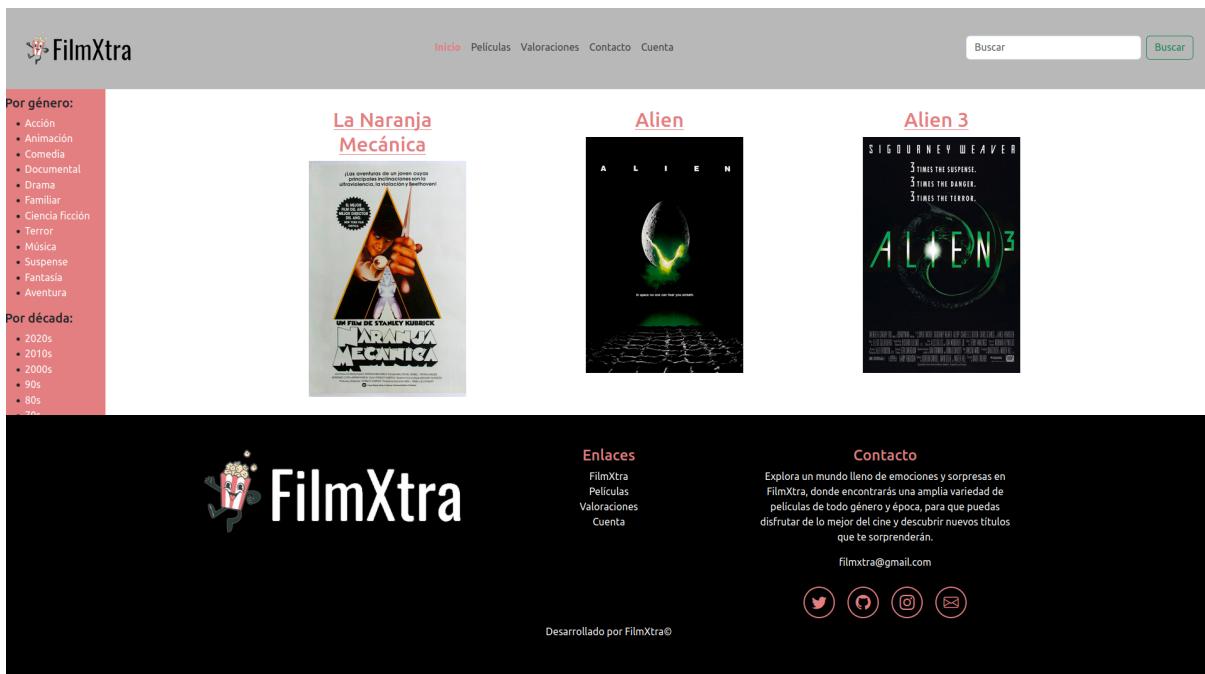


Durante la creación de esta página, nos encontramos con cierta confusión inicial. Al principio, teníamos la idea de implementar código de Bootstrap en Laravel. Elena trabajó en una primera versión de la página de inicio, utilizando HTML5 y CSS3 con Bootstrap. Sin embargo, más adelante descubrimos que Laravel tenía su propio framework por defecto llamado Tailwind basado en Vue.js.

Ante este descubrimiento, nos dimos cuenta de que debíamos adaptarnos y utilizar el framework de Tailwind en lugar de Bootstrap. Esto significó que Elena tuvo que aprender rápidamente cómo programar con Tailwind y rehacer la página de inicio en poco tiempo.

A pesar de este contratiempo, Elena se enfrentó al desafío y logró crear una nueva versión de la página de inicio utilizando el framework de Tailwind. Aprendió rápidamente las peculiaridades de este framework y adaptó el código para asegurarnos de que estábamos siguiendo las mejores prácticas y utilizando las herramientas adecuadas proporcionadas por Laravel.

Esta experiencia nos enseñó la importancia de estar abiertos a la adaptación y a aprender nuevas herramientas, incluso cuando ya hemos avanzado en el proceso de desarrollo. Aprendimos a ser flexibles y a aprovechar al máximo las capacidades y las herramientas predeterminadas que ofrece Laravel, como el framework de Tailwind, para garantizar un desarrollo eficiente y coherente en nuestro proyecto.



Debido a este obstáculo, se presentó un nuevo diseño que es el que actualmente se encuentra en la web. Elena trabajó en el código utilizando HTML5, CSS3 y Tailwind, y luego todo ese código se clasificó en componentes. Este enfoque nos permitió tener una buena estructuración de todas las páginas y reutilizar el código de manera eficiente.

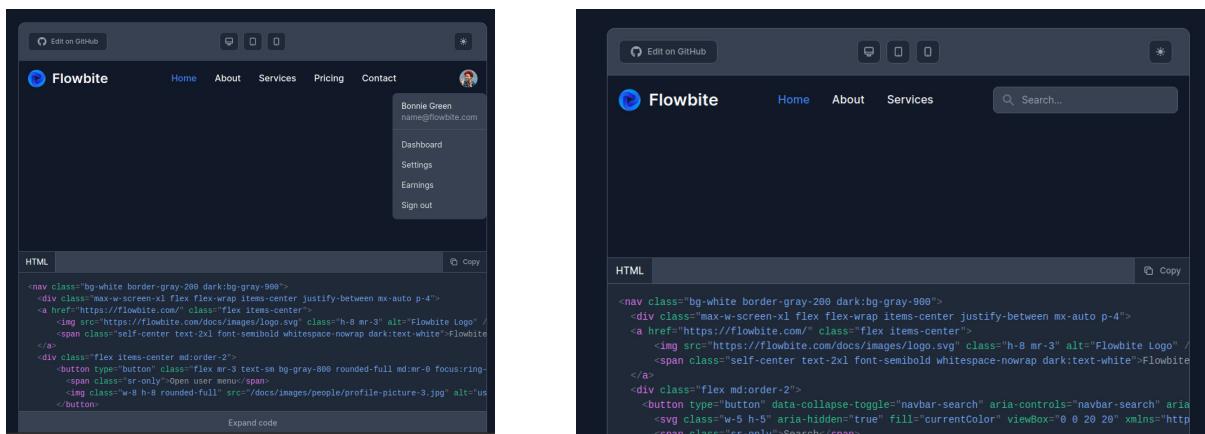
En la página de inicio, denominada "Welcome.vue" en nuestro proyecto, utilizamos tres componentes principales:

- **LayoutPrincipal:** el "Layout principal" es el diseño utilizado en todas las páginas, excepto en aquellas que no requieren una barra de navegación o un pie de página. Este layout consta de una barra superior y un pie de página, y se mantiene constante sin actualizarse con cada carga de página. Por ello, permite incrustar elementos que se mantienen entre las recargas de las páginas.

Creamos este layout como un componente para evitar la duplicación innecesaria de código, ya que estos elementos se utilizan en la mayoría de las páginas. La barra de navegación se encuentra en la parte superior y muestra el logo de FilmXtra, cuatro enlaces, un buscador y una sección para funciones de inicio de sesión, registro, cierre de sesión y acceso a la cuenta del usuario. El pie de página consta del logo de FilmXtra y dos columnas responsivas. En pantallas de ordenador y tablet, las columnas se muestran junto al logo, mientras que en dispositivos móviles se colocan debajo del logo. Además, se incluye un span y un div que contienen una frase y los iconos de las redes sociales.

Para crear la barra de navegación y el pie de página, nos basamos en la biblioteca gratuita de Tailwind llamada Flowbite, ya que los componentes que ofrece Tailwind son de pago. Tomamos ejemplos de Flowbite y los adaptamos a nuestro estilo, colores y diseño, logrando una barra de navegación y un pie de página que se ajustan perfectamente a nuestras necesidades.

#### Modelos para la barra de navegación:



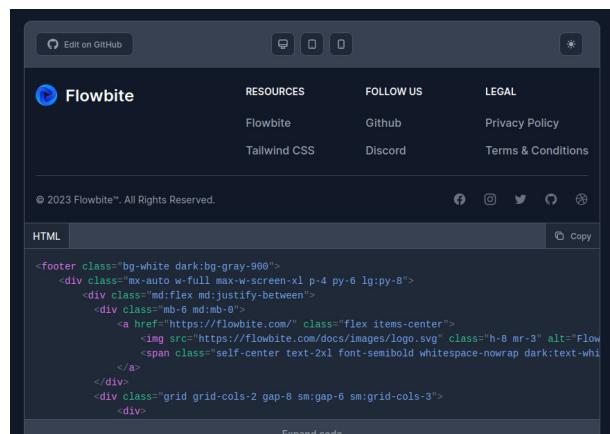
The screenshot shows two versions of a navigation bar component from the Flowbite library. The left version is a dark-themed navigation bar with a user profile dropdown menu open, showing options like Dashboard, Settings, Earnings, and Sign Out. The right version is a similar navigation bar but with a search bar integrated into the top right corner. Both versions include an 'Edit on GitHub' button at the top left and a 'Copy' button at the top right. Below each screenshot is a snippet of the generated HTML code.

```
<nav class="bg-white border-gray-200 dark:bg-gray-900">
  <div class="max-w-screen-xl flex flex-wrap items-center justify-between mx-auto p-4">
    <a href="https://flowbite.com/" class="flex items-center">
      
      <span class="self-center text-2xl font-semibold whitespace nowrap dark:text-white">Flowbite</span>
    </a>
    <div class="flex items-center md:order-2">
      <button type="button" class="flex er-3 text-sm bg-gray-800 rounded-full md:mr-0 focusing">
        <span class="sr-only">Open user menu</span>
        
      </button>
    </div>
  </div>
</nav>
```

```
<nav class="bg-white border-gray-200 dark:bg-gray-900">
  <div class="max-w-screen-xl flex flex-wrap items-center justify-between mx-auto p-4">
    <a href="https://flowbite.com/" class="flex items-center">
      
      <span class="self-center text-2xl font-semibold whitespace nowrap dark:text-white">Flowbite</span>
    </a>
    <div class="flex md:order-2">
      <button type="button" data-collapse-toggle="navbar-search" aria-controls="navbar-search" aria-expanded="false" class="text-gray-400 dark:text-gray-500 focus:outline-none focus:ring-2 focus:ring-indigo-500 focus:ring-offset-2" data-mdb-order="2">
        <span class="sr-only">Search</span>
        <svg class="w-5 h-5" aria-hidden="true" fill="currentColor" viewBox="0 0 20 20" xmlns="http://www.w3.org/2000/svg"></svg>
      </button>
    </div>
  </div>
</nav>
```

#### Y modelos para el pie de página:



The screenshot shows a footer component from the Flowbite library. It features a dark background with white text and icons. The footer is divided into three main sections: 'RESOURCES' (links to Flowbite and Tailwind CSS), 'FOLLOW US' (links to GitHub and Discord), and 'LEGAL' (links to Privacy Policy and Terms & Conditions). At the bottom, there's a copyright notice for 2023 Flowbite. Below the footer is a snippet of the generated HTML code.

```
<footer class="bg-white dark:bg-gray-900">
  <div class="mx-auto w-full max-w-screen-xl p-4 py-6 lg:py-8">
    <div class="md:flex md:justify-between">
      <div class="mb-6 md:mb-0">
        <a href="https://flowbite.com/" class="flex items-center">
          
          <span class="self-center text-2xl font-semibold whitespace nowrap dark:text-white">Flowbite</span>
        </a>
      </div>
      <div class="grid grid-cols-2 gap-8 sm:gap-6 sm:grid-cols-3">
        <div>
```

Es importante resaltar que, si bien nos inspiramos en ejemplos existentes, realizamos las adaptaciones necesarias para que la barra de navegación y el pie de página se ajustarán a nuestras necesidades y reflejarán la identidad visual de FilmXtra. Además, nos enfocamos en crear componentes responsivos que ofrecieran una experiencia de usuario óptima en diversos dispositivos y tamaños de pantalla. Por lo tanto, tanto la barra de navegación como el pie de página se presentan en tres versiones diferentes: ordenador, tablet y móvil.

The figure displays three versions of the FilmXtra website's navigation bar:

- Desktop Version:** Shows a top navigation bar with links for 'Inicio', 'Top FilmXtra', 'Top Valoraciones', 'Cuenta', a search icon, and a user profile icon.
- Tablet Version:** Shows a top navigation bar with links for 'Inicio', 'Top FilmXtra', 'Top Valoraciones', 'Cuenta', a search bar containing 'Buscar...', and a user profile icon.
- Mobile Version:** Shows a top navigation bar with links for 'Inicio', 'Top FilmXtra', 'Top Valoraciones', 'Cuenta', a search icon, and a menu icon (three horizontal lines). Below this is a main navigation bar with links for 'Inicio', 'Top FilmXtra', 'Top Valoraciones', and 'Cuenta'.

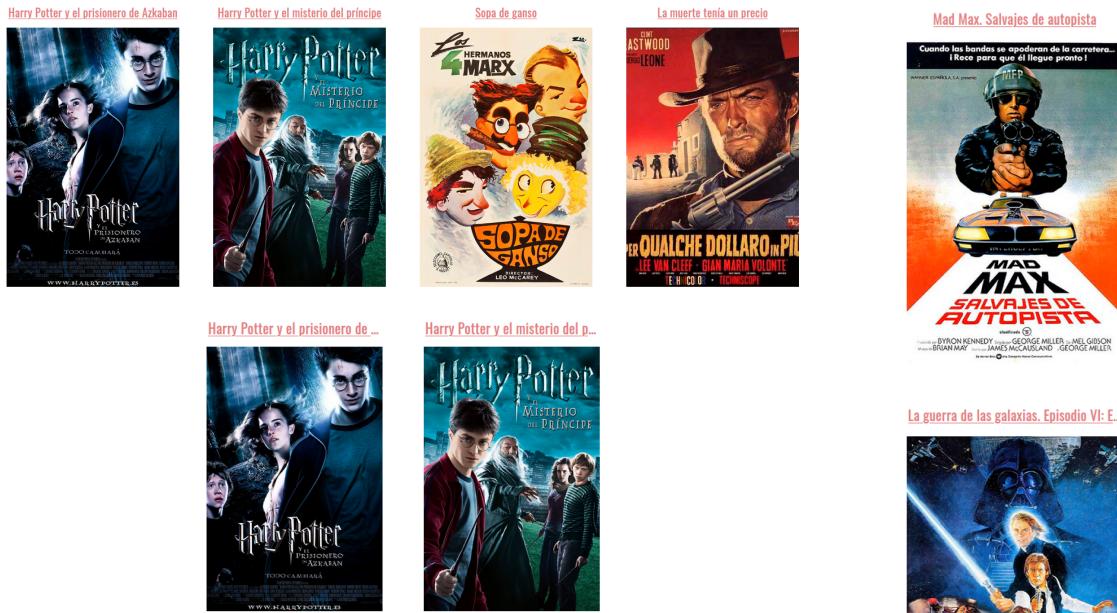
Below these are two footer sections:

- Left Footer (Dark Mode):** Includes links for 'ENLACES' (FilmXtra, Top FilmXtra, Top Valoraciones, Cuenta), 'CONTACTO' (filmxtra.23@gmail.com), a descriptive paragraph about the site's mission, and social media links.
- Right Footer (Dark Mode):** Includes links for 'ENLACES' (FilmXtra, Top FilmXtra, Top Valoraciones, Cuenta), 'CONTACTO' (filmxtra.23@gmail.com), a descriptive paragraph about the site's mission, and social media links.

- **Carrusel:** este componente muestra un carrusel con cinco imágenes que se reproducen automáticamente al abrir la web. Además, proporciona flechas y puntos que permiten la navegación manual entre las imágenes. Debido a que su objetivo es agregar dinamismo y atractivo visual a la página de inicio.



- Póster:** este último componente muestra dieciséis pósters con sus respectivos títulos en filas. Se seleccionan aleatoriamente y se distribuyen en columnas, al igual que el pie de página. En pantallas grandes, se muestran cuatro pósters por fila, en tablets se muestran dos y en dispositivos móviles se muestra uno. De esta manera, la página de inicio es totalmente responsive y se adapta a cualquier dispositivo.

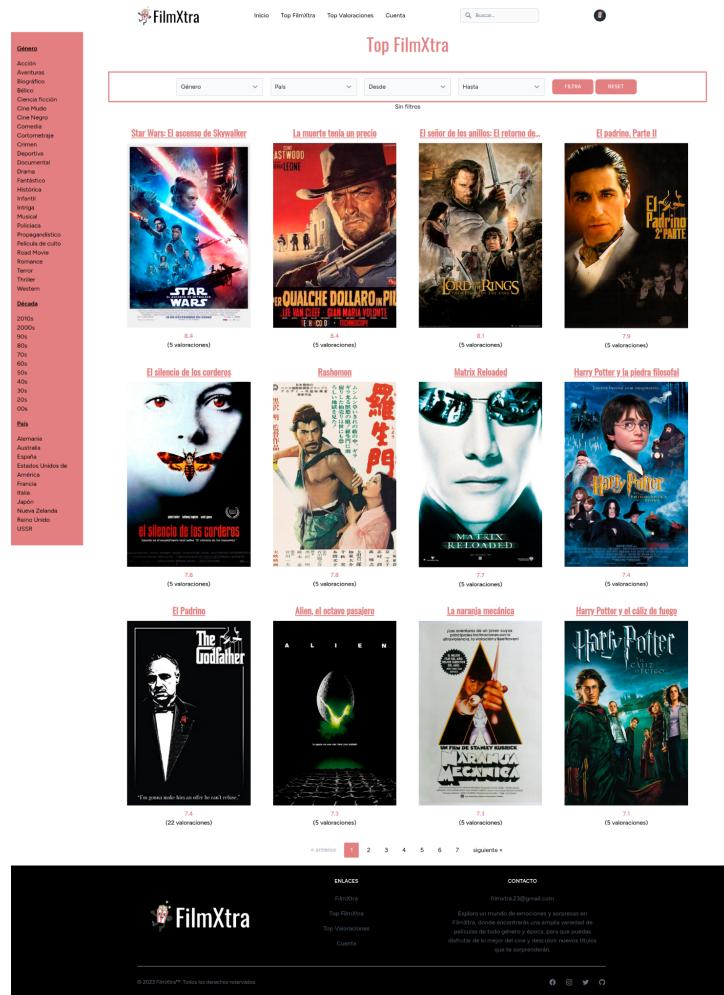


### 3.2. Página "Top"

La página "Top" se muestra cuando se hace clic en el enlace "Top FilmXtra" de la barra de navegación. A lo largo de la evolución de los prototipos, esta página, al igual que la de inicio, ha experimentado pocos cambios significativos. En todas las versiones, se mantiene el layout principal, que incluye la barra de navegación y el pie de página. Además, cuenta con una barra lateral que muestra los géneros, épocas y países disponibles en nuestra base de datos. Y, en columnas de cuatro, se presentan las películas con su póster y título identificativo.

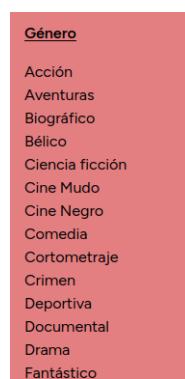
En la versión final, se agregó un componente de filtrado que no estaba presente en ninguno de los prototipos anteriores. Esta idea fue desarrollada por Víctor, ya que inicialmente se planeaba utilizar un desplegable para elegir películas por música, animación, ordenar por año o por orden alfabético. Sin embargo, con el tiempo, esta opción no nos convenció y, con la ayuda de Víctor, Elena implementó un filtro más completo que complementa el filtrado de la barra lateral.

El diagrama muestra la evolución del diseño de la página de películas. A la izquierda, un boceto titulado 'PELÍCULAS Y SERIES' muestra una estructura básica con secciones para 'Por género', 'Por época' y 'Por país', y un cuadro central para 'Películas y series'. Un gran flecha apunta hacia la derecha, donde se muestra el diseño final: 'PÁGINA DE PELÍCULAS'. El diseño incluye una barra superior con 'LOGO', 'Películas', 'Valoraciones', 'Buscar...', 'Mi cuenta' y un menú desplegable. Una barra lateral izquierda enumera 'Por género', 'Por época' y 'Por país'. La sección central muestra una lista de películas con imágenes y títulos, como 'La Llorona', 'Grease', 'Mamma mia!', 'Elvis' y 'Yesterday'. Bajo la lista hay secciones para 'Socialízate' y 'Mi cuenta Películas Valoraciones'. Al pie de la página, se incluye el texto '© 2023. Desarrollado por Nombre grupo.'



Al igual que en las páginas anteriores, esta también está compuesta por varios componentes, que son los siguientes:

- **LayoutPrincipal:** este componente engloba todos los demás componentes de la página, como en las páginas anteriores. Proporcionando así, la estructura general de la página.
- **BarraLateral:** la barra lateral muestra los datos relacionados con género, época y país que se encuentran en nuestra base de datos. Al hacer clic en alguno de estos campos, se muestra la información correspondiente de forma dinámica. Además, el formulario de filtrado de la barra lateral permite al usuario seleccionar filtros específicos que se reflejan en el campo inferior del formulario de filtrado, en tiempo real.
- **FormularioFiltrado:** el formulario de filtrado diseñado por Víctor está compuesto por cuatro selects que muestran la información disponible de nuestra base de datos, como los géneros de películas, países y años. El formulario también incluye dos botones: "Filtrar" para confirmar los filtros seleccionados por el usuario, y "Reset" para eliminar todos los filtros. Además, cuenta con un campo de texto que nos indica los filtros aplicados en el momento, por lo que si por ejemplo realizas una búsqueda de películas de ciencia ficción estadounidenses, en este campo pondrá "Filtros: género: Ciencia Ficción, país: Estados Unidos de América."



FILTRA
RESET

Sin filtros

- **Póster:** este componente es similar al utilizado en las páginas de inicio y obra. Muestra doce pósters con sus respectivos títulos distribuidos en filas. Dependiendo del tamaño de la pantalla, se mostrarán cuatro pósters por fila en pantallas grandes, dos en tablets y uno en dispositivos móviles. Por lo que el componente se adapta de manera responsiva a diferentes dispositivos.
- **Paginación:** y por último, la paginación permite navegar entre las páginas de pósters. Se muestran doce películas por página, excepto en la última página, que puede contener menos películas. La paginación se realiza sin recargar toda la página, actualizando únicamente el componente que muestra los pósters y títulos correspondientes.

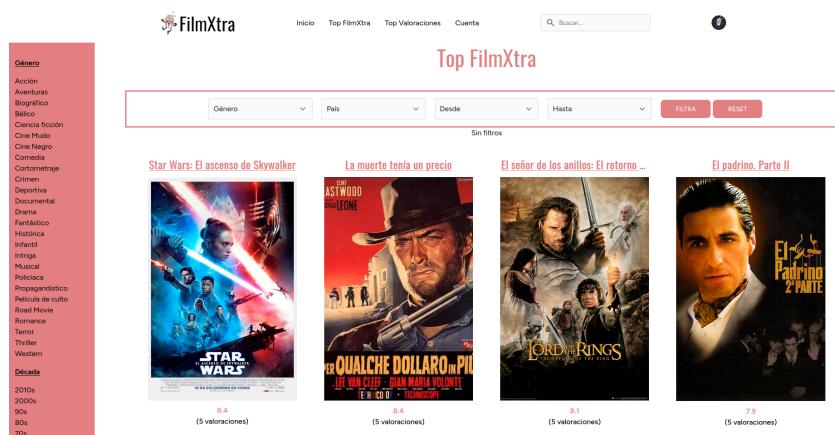
« anterior      1      2      3      4      5      6      7      siguiente »

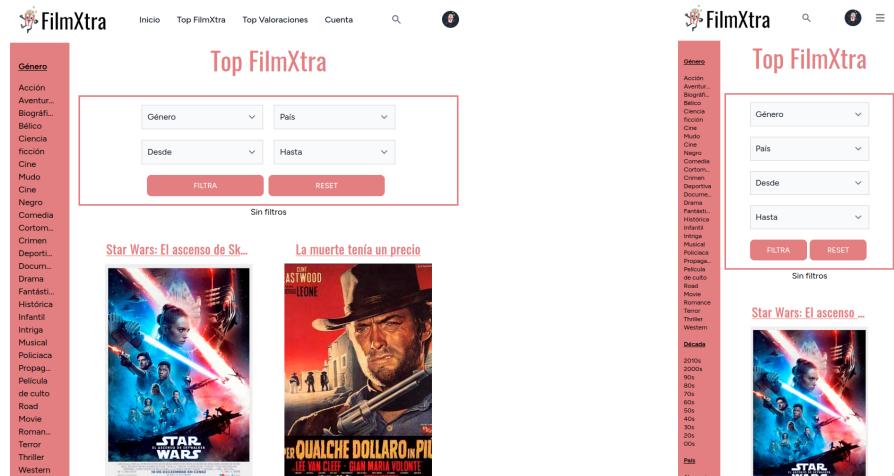
La página "Top" está compuesta por diferentes componentes que se adaptan de forma responsable a distintos tamaños de pantalla. Veamos en detalle cómo se compone y cómo se logra su responsividad.

En primer lugar, el diseño de la página ha sido configurado para que todos los elementos se adapten y se muestren correctamente en pantallas más pequeñas. Un ejemplo de ello es la barra lateral, la cual está contenida en un div que utiliza tecnología flex. Esto permite que los elementos del contenedor se coloquen en un formato flexible y en línea. Además, se ha utilizado el flex-wrap de Tailwind para que, al reducirse el tamaño de la pantalla, los elementos salten automáticamente y se coloquen uno debajo del otro. En cuanto al tamaño de la barra lateral, en pantallas grandes ocupa un ancho de 10 Viewport Width (VW), equivalente aproximadamente al 10% del ancho visible de la pantalla. En pantallas de tablet, su ancho es de 13 VW, y en dispositivos móviles es de 20 VW. De esta manera, la barra lateral se ajusta y se muestra correctamente en diferentes dispositivos.

El contenido principal de la página se encuentra a la derecha de la barra lateral. Aquí se inserta el formulario de filtrado, que permite al usuario refinar su búsqueda seleccionando género, país y rango de años. Este contenido se adapta al espacio restante de la pantalla y se muestra de manera adecuada en todos los dispositivos.

A continuación, al igual que en la página de inicio, se muestra una sucesión de cuatro películas por fila. Cada película se presenta con su póster y título correspondiente, y están ordenadas según su nota de valoración. En total, se muestran doce películas por página. Sin embargo, gracias al componente de "Paginación" desarrollado por nuestro compañero Víctor, es posible mostrar más películas sin tener que recargar toda la página. Esto se logra actualizando únicamente el componente que muestra los pósters y los títulos de las películas, lo que mejora significativamente la velocidad y la experiencia de navegación del sitio web.





Con esta estructura y su diseño responsive, la página "Top" se adapta de forma óptima a diversos tamaños de pantalla, brindando una experiencia de usuario consistente y eficiente.

### 3.3. Página "Obra"

La página de "Obra" es la que se muestra al hacer clic en un póster o título específico para obtener información detallada. Aunque no se creó un prototipo en papel para esta página, podemos observar su evolución a través del prototipo en Figma.

Inicialmente, la idea de Elena era mostrar el póster en un lateral, seguido de toda la información disponible en la base de datos, y luego mostrar las críticas de los usuarios y el apartado para valorar personalmente con una calificación y comentario. Sin embargo, posteriormente se realizaron cambios en la implementación del backend y frontend, dividiendo la página en dos: una para consultar los datos y otra para realizar valoraciones. Además de esto, se planeaba añadir el trailer de la película consultada como último elemento en esta página.

The diagram illustrates the evolution of the movie detail page through three stages:

- PÁGINA DE UNA PELÍCULA EN CONCRETO (Desktop View):** Shows a main content area with a movie poster for "Marte (The Martian)", basic movie details, and a trailer section. Below is a footer with social media links and a copyright notice.
- El Padrino (Desktop View):** Shows a detailed movie page for "The Godfather". It includes a large poster, user reviews, and a sidebar with movie facts. A large black arrow points from the first stage to this one.
- El Padrino (Mobile View):** Shows the same movie detail page as the previous stage, but optimized for a smaller screen, demonstrating responsive design.

Esta página cuenta con cuatro componentes principales:

- **LayoutPrincipal:** incluye la barra de navegación y el pie de página, proporcionando la estructura y el diseño general de la página.
- **Estrellitas:** este componente se encuentra debajo de cada póster y está compuesto por diez estrellas. La cantidad de estrellas que se rellenan depende de la valoración de la película. Por ejemplo, si la película tiene una valoración de 5,3 se llenarán cinco estrellas, y si tiene una valoración de 8,8 se llenarán nueve estrellas. La valoración se redondea, y el número resultante determina cuántas estrellas se mostrarán.



- **Póster:** al igual que en la página de inicio, en la página "Obra" se muestra el póster correspondiente de cada película.
- **Trailer:** este componente recupera de la base de datos el iframe embeddedo insertado en la tabla "trailers" y lo muestra en la página. Cada película, identificada por un número único, tiene su propio trailer asociado en la tabla "trailers". De esta manera, se puede visualizar el trailer correspondiente a la película consultada.



Una vez explicados los componentes principales de la página, pasemos a detallar su creación y su adaptabilidad a diferentes dispositivos.

En primer lugar, se creó un contenedor <div> que contiene el título de la película, el póster y su información. El póster y la información se encuentran dentro de otro <div> con la clase "grid", que divide el contenido en columnas. En pantallas de ordenador y tablet, se muestran en dos columnas, con el póster y la información uno al lado del otro. En cambio, en dispositivos móviles, se configuró para que se muestren en una sola columna, comenzando con el título, seguido del póster y, a continuación, la información.

Debajo del póster, siempre se muestra el componente "estrellitas" seguido de la valoración de la película sobre diez.

Además, en la sección de información, se puede ver que si la película tiene una secuela, precuela o spin-off, se muestra el póster y el título de esa película relacionada.

A continuación, se muestra un contenedor de críticas. Este <div> también está estructurado en columnas para adaptarse a diferentes tamaños de pantalla. En pantallas de ordenador, se divide en cuatro columnas, en tablet en tres columnas, y en dispositivos móviles en una columna. Sin embargo, dado que hay dos secciones dentro de este contenedor (una para mostrar las críticas y otra para explicar las instrucciones para valorar), cuando se visualiza en ordenadores y tablets, el apartado de críticas profesionales ocupa tres columnas en ordenadores y dos columnas en tablets, aprovechando el espacio restante en el lado derecho. Y en dispositivos móviles, se muestra en una sola columna.

**Criticas profesionales:**

- Sensacione: El padrino son palabras mayores. Las dos primeras partes están entre las 10 mejores películas de la historia del cine. Víctor G. (hace 51 años)
- Super Hero Hype: Coppola inventa una nueva mirada para el cine y amplía los horizontes de una industria que pedía a gritos savia nueva. Cristina B. (hace 51 años)
- Box Office Mojo: Brando hizo de Don Vito algo que rara vez vemos en las películas: un villano-héroe tragicómico. Raquel B. (hace 50 años)
- Rotten Tomatoes: La secuencia de la boda es un momento cinematográfico virtuoso: Coppola lleva a su gran reparto con tanta destreza que nos hace entrar completamente en el mundo de El Padrino. Víctor G. (hace 51 años)
- La Vanguardia: Una obra de arte que perdura el paso de los años. Noa Z. (hace 51 años)

**Criticas de nuestros usuarios:**

- Luisa: La mafia siempre triunfa, una muy buena película. La podríamos clasificar como un clásico del cine, un indispensable en la lista de películas que ver. (hace 17 horas) - Likes: 25
- Tom: Una película de 10, de las que ya no hacen. (hace 17 horas) - Likes: 24
- ...

[Ver más críticas de usuarios/Valorar El Padrino→](#)

**¿Quieres valorar esta película?**

En FilmXtra nos apasiona el cine y queremos escuchar tu voz. ¡Expresate como quieras!

Puedes ponerle una puntuación del 1 al 10 a las películas que ves.

Si te gusta entrar en detalles, déjanos tus críticas más elaboradas. ¡Suelta todo lo que piensas!

Y, por supuesto, dales un buen "like" a las críticas de otros usuarios que te parezcan geniales. ¡Comparte el amor cinéfilo!

[Top Valoraciones →](#)

**Criticas profesionales:**

- Sensacione: El padrino son palabras mayores. Las dos primeras partes están entre las 10 mejores películas de la historia del cine. Víctor G. (hace 51 años)
- Super Hero Hype: Coppola inventa una nueva mirada para el cine y amplía los horizontes de una industria que pedía a gritos savia nueva. Cristina B. (hace 51 años)
- Box Office Mojo: Brando hizo de Don Vito algo que rara vez vemos en las películas: un villano-héroe tragicómico. Raquel B. (hace 50 años)
- Rotten Tomatoes: La secuencia de la boda es un momento cinematográfico virtuoso: Coppola lleva a su gran reparto con tanta destreza que nos hace entrar completamente en el mundo de El Padrino. Víctor G. (hace 51 años)
- La Vanguardia: Una obra de arte que perdura el paso de los años. Noa Z. (hace 51 años)

**Criticas de nuestros usuarios:**

- Luisa: La mafia siempre triunfa, una muy buena película. La podríamos clasificar como un clásico del cine, un indispensable en la lista de películas que ver. (hace 17 horas) - Likes: 25
- Tom: Una película de 10, de las que ya no hacen. (hace 17 horas) - Likes: 24
- ...

Por último, como se mencionó anteriormente, se muestra el trailer de la película en cuestión.



### 3.4. Página "Top Valoraciones"

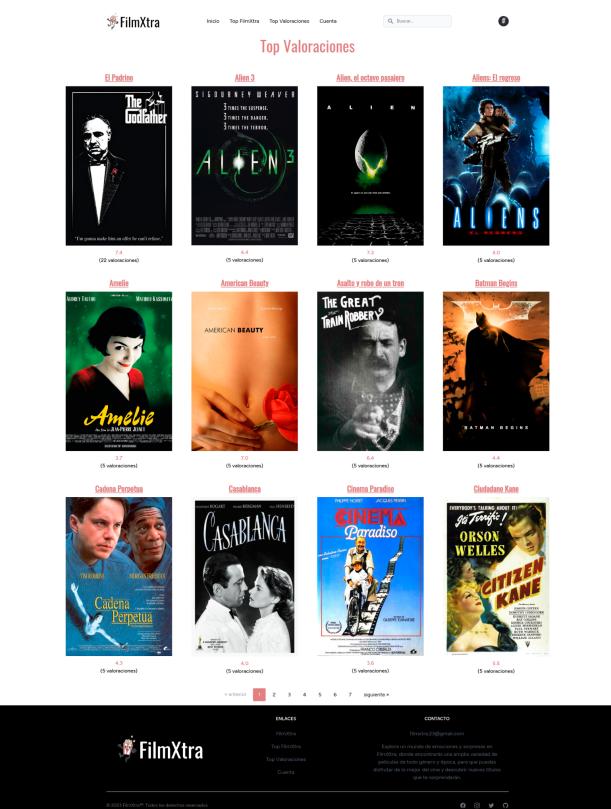
La página de "Top Valoraciones" está diseñada específicamente para permitir a los usuarios valorar y criticar películas, al igual que la página de "Top FilmXtra". Esta página está compuesta por tres componentes principales.

- **LayoutPrincipal:** este componente es fundamental, ya que engloba a todos los demás y proporciona la barra de navegación y el pie de página. Su función principal es establecer la base para el diseño y la navegación en toda la página.
- **Póster:** en este componente, se muestran doce películas con sus respectivos pósters y títulos, distribuidos en filas. La cantidad de pósters por fila varía según el tamaño de la pantalla. En pantallas grandes se mostrarán cuatro pósters por fila, en tablets dos, y en dispositivos móviles uno. De esta manera, el componente se adapta de forma responsiva a diferentes dispositivos para brindar la mejor experiencia visual posible.

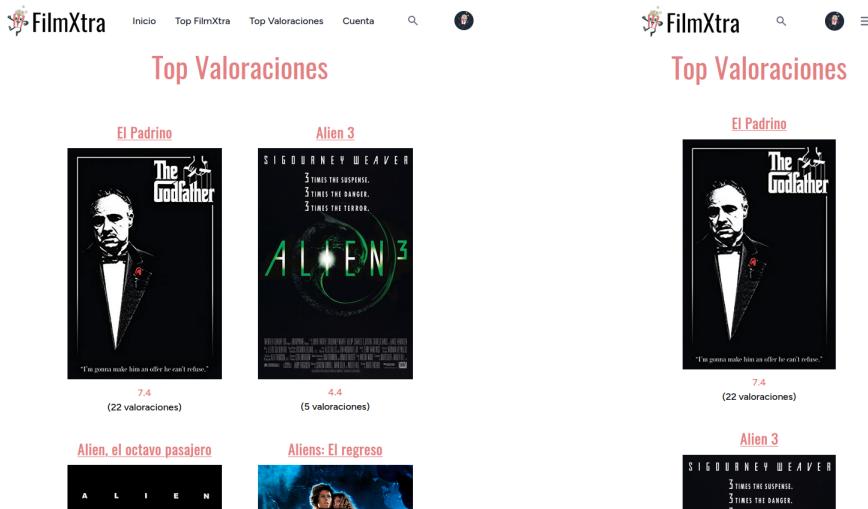
Cuando se hace clic en uno de los pósters, en lugar de ver la información de la película, los usuarios pueden leer las valoraciones de otros usuarios. Tienen la opción de dar "me gusta" a esas valoraciones y, si lo desean, pueden expresar su propia opinión personal sobre la película, otorgando una calificación del cero al diez o escribiendo una crítica de hasta cinco mil caracteres. Sin embargo, todas estas acciones solo están disponibles para los usuarios que hayan creado una cuenta y hayan iniciado sesión.

- **Paginación:** este componente permite la navegación entre las diferentes páginas de pósters. Cada página muestra doce películas, excepto la última página, que puede contener menos películas. La paginación se realiza sin tener que recargar toda la página, actualizando únicamente el componente que muestra los pósters y títulos correspondientes. Esto agiliza la navegación y mejora la velocidad de la página.

Con esta combinación de componentes, la página de "Top Valoraciones" ofrece a los usuarios la posibilidad de explorar películas, leer y dar "me gusta" a las valoraciones de otros usuarios, y compartir sus propias opiniones y críticas. Todo ello en un entorno responsive y fácil de navegar.



La pantalla de "Top Valoraciones" fue diseñada y creada sin un prototipo previo en papel o Figma. Al observar la imagen anterior, se puede apreciar que tiene un aspecto similar al de "Top FilmXtra". Sin embargo, dado que no tenía sentido incluir la barra lateral y el formulario de filtrado en esta página, se decidió utilizar únicamente el componente Póster para mostrar las películas, siguiendo el mismo diseño y responsividad que en "Top FilmXtra".



### 3.5. Página "Ficha Valoraciones"

Para la página de "Ficha Valoraciones", el diseño se desarrolló de manera iterativa, basándonos en la página de "Obra" y manteniendo la coherencia visual y minimalista en toda la web. La idea era que la página siguiera el mismo estilo serio y minimalista, sin importar si se accedía desde "Inicio", "Top FilmXtra" o "Top Valoraciones".

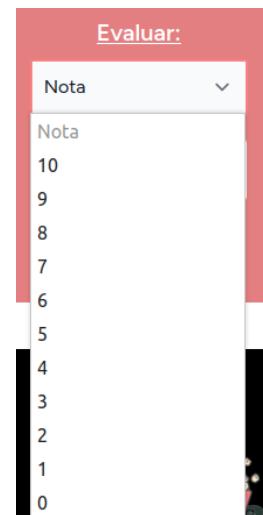
A continuación, describiremos los componentes principales que conforman esta página:

- **LayoutPrincipal:** este componente es el elemento principal que contiene la barra de navegación y el pie de página, y sirve como estructura base para todos los demás componentes.
- **Estrellitas:** como mencionamos anteriormente en la página de "Obra", este componente se encuentra debajo de cada póster y consiste en un conjunto de diez estrellas. La cantidad de estrellas que se llenan depende de la valoración de la película. Por ejemplo, si una película tiene una valoración de 5.3, se llenarán cinco estrellas; si tiene una valoración de 8.8, se llenarán nueve estrellas. La valoración se redondea y el número resultante determina cuántas estrellas se mostrarán.
- **Paginación:** al igual que en páginas anteriores, este componente permite la navegación entre diferentes páginas. En este caso, se implementa para las críticas de los usuarios. Se muestran hasta cuatro críticas por página, pero si una película no cuenta con suficientes críticas para completar una página, se mostrarán las disponibles.

- **SelectRango:** básicamente, esta funcionalidad en el backend permite establecer límites superiores e inferiores basados en consultas a una base de datos MySQL. Por ejemplo, si realizas una consulta para obtener los años de las películas existentes, puedes obtener el año más antiguo y el más reciente. Utilizando esos valores como límites, se generan dinámicamente las opciones disponibles.

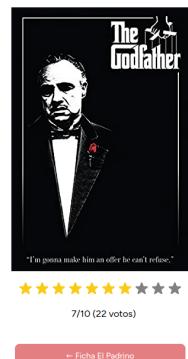
Esto se logra pasando los límites como props desde el backend al componente correspondiente. Los límites, por ejemplo, pueden ser utilizados para generar las opciones en un elemento select. De esta manera, el usuario tendrá la posibilidad de seleccionar un rango específico de años.

En resumen, esta funcionalidad permite adaptar las opciones disponibles en función de los datos obtenidos de la base de datos, permitiendo una selección más precisa por parte del usuario.



Una vez hemos documentado los componentes que componen la página de "Ficha Valoraciones", pasamos a hablar sobre su diseño y elaboración.

Nuestro objetivo era que esta página tuviera una apariencia similar a la de "Obra", por ello, al abrir la página, lo primero que se muestra es el póster de la película junto con su calificación, tanto en formato de estrellas como en formato numérico, que refleja todos los votos que han contribuido a esa calificación. Justo debajo, se encuentra un botón que dice "Ficha" seguido del nombre de la película en cuestión. Si se hace clic en este botón, se redirige al usuario a la página de "Obra", donde se puede consultar toda la información relacionada con la película.



A la derecha de esta sección, encontramos un apartado que muestra todas las críticas de los usuarios presentes en nuestra base de datos. Gracias al componente de "Paginación", en el caso de películas con muchas críticas, se pueden mostrar hasta cuatro críticas por página. Sin embargo, si no hay suficientes críticas para llenar una página, se muestran todas las disponibles.

Además, para las críticas utilizamos una librería de JavaScript para el manejo de fechas en las páginas de "Obra" y "FichaValoración". Esta librería nos permite mostrar fechas relativas junto a las críticas. La instalamos a través de npm, un gestor de paquetes que forma parte de nuestras tecnologías utilizadas. Con esta implementación, buscamos mejorar la experiencia de los usuarios al brindarles información relevante sobre la temporalidad de las valoraciones.

- **Críticas de nuestros usuarios:**

- Luisa: La mafia siempre triunfa, una muy buena película. La podríamos clasificar como un clásico del cine, un indispensable en la lista de películas que ver. (hace un día) - Likes: 25 |
- Tom: Una película de 10, de las que ya no hacen. (hace un día) - Likes: 24 |

Toda esta estructura se encuentra dentro de un div que se divide en cuatro columnas en el diseño de ordenador y una columna en el diseño de tablet y móvil. En el diseño de ordenador, el div que contiene las críticas tiene una clase llamada "col-span-3", lo que hace que esta sección ocupe tres de las cuatro columnas disponibles, dejando la columna restante para el póster y la calificación.

The screenshot shows the FilmXtra website's movie detail page for 'El Padrino'. At the top right is the FilmXtra logo and navigation links: Inicio, Top FilmXtra, Top Valoraciones, Cuenta, and a search bar. Below the header is a large movie poster for 'The Godfather'. To the left of the poster is a sidebar with a smaller image of Marlon Brando and the quote '“Tu papá nació bien, no es él el que está mal.”'. Below this is a 5-star rating with the text '7/10 (22 votos)'. The main content area has a red header 'El Padrino'. Underneath it is a box titled '• Críticas de nuestros usuarios:' containing two reviews from 'Luis' and 'Tom'. To the right of the reviews is a large image of Marlon Brando in a tuxedo with a bow tie, with the quote '“Tu papá nació bien, no es él el que está mal.”' at the bottom. Below the image is a 5-star rating and the text '7/10 (22 votos)'. A red button at the bottom right says 'Ir a la ficha de El Padrino'.

Por último, tenemos otro div dividido en cuatro columnas en el diseño de ordenador y una columna en el diseño de tablet y móvil. En este div se encuentra un formulario donde los usuarios registrados y logueados pueden calificar la película en una escala del cero al diez. Si es la primera vez que califican la película, se muestra un mensaje y un GIF, mientras que si modifican su calificación, el mensaje y el GIF cambiarán. Los GIFs se encuentran en la carpeta "/public" y se muestran de forma aleatoria.

En cuanto al segundo apartado de este div, permite a los usuarios registrados y logueados dejar una reseña de hasta cinco mil caracteres. Al igual que en el apartado anterior, si un usuario realiza su primera crítica a una película, se muestra un mensaje y un GIF, pero si modifican su reseña, el mensaje cambiará y se mostrará otro GIF.

Todas estas funcionalidades se muestran únicamente si el usuario ha iniciado sesión. Si no, se mostrará un mensaje indicando que es necesario hacerlo para acceder a esas secciones, como se puede observar en la foto tres.

The top screenshot shows a user who is logged in, able to evaluate a movie. It displays a rating form with a dropdown menu set to 'Nota' and a button 'Evaluar El Padrino'. To the right is a large text input field for a review with the placeholder 'Reseña El Padrino (0/5000 caracteres)' and a 'Resellar El Padrino' button below it. The bottom screenshot shows a message for users who are not logged in: 'Para poder evaluar o poner notas a la película tienes que estar logueado.' It includes a 'Loguearse' button and the FilmXtra logo.

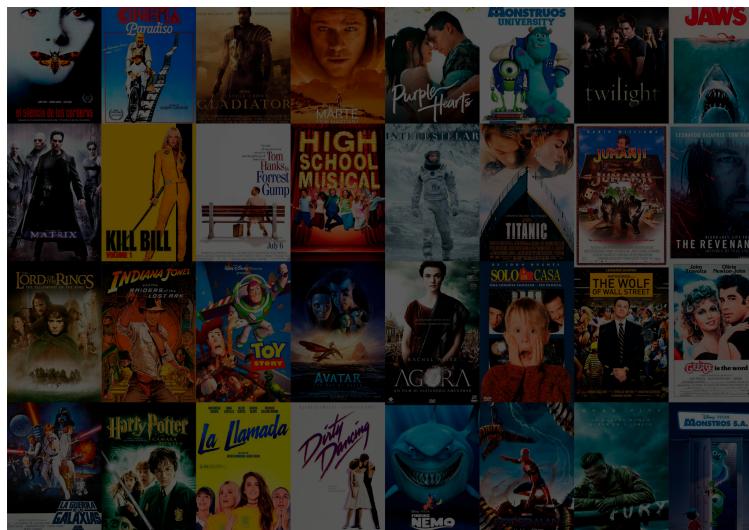
### 3.6. Página "Login" y "Register"

Para las páginas de inicio de sesión, registro y otras destinadas a los usuarios, decidimos adoptar un enfoque distinto al no utilizar el componente "LayoutPrincipal", gracias a una idea propuesta por Elena después de investigar sitios web dedicados a la valoración de películas.

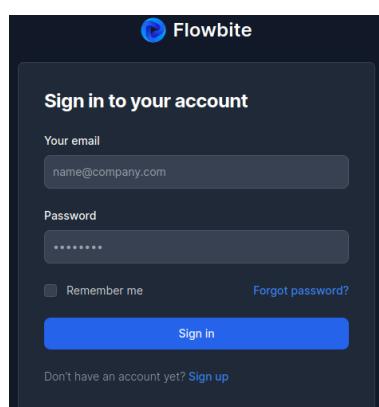
Elena se inspiró en el diseño de estos sitios y se le ocurrió una forma creativa de darles un aspecto único. Para lograrlo, recopiló todos los pósters que habíamos recortado previamente, y también descargó y recortó algunos otros, y creó un fondo de pantalla utilizando estos pósters.

Esta decisión fue tomada con el objetivo de diferenciar claramente estas páginas como espacios exclusivamente dedicados a los usuarios, en contraste con aquellas páginas donde se consulta información o se leen críticas.

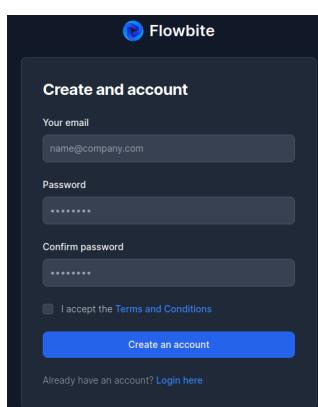
El uso de este fondo de pantalla personalizado agrega un toque distintivo y ayuda a los usuarios a reconocer rápidamente cuando están accediendo a una página destinada a funciones específicas de usuario, como inicio de sesión y registro.



Para las páginas de "[Login](#)" y "[Register](#)", Elena utilizó componentes predefinidos de Tailwind como base para su creación. Sin embargo, como es habitual, personalizó el aspecto de estos componentes para que se alinearan con la identidad visual de nuestra marca, FilmXtra. También adaptó los formularios para incluir los campos de datos relevantes.



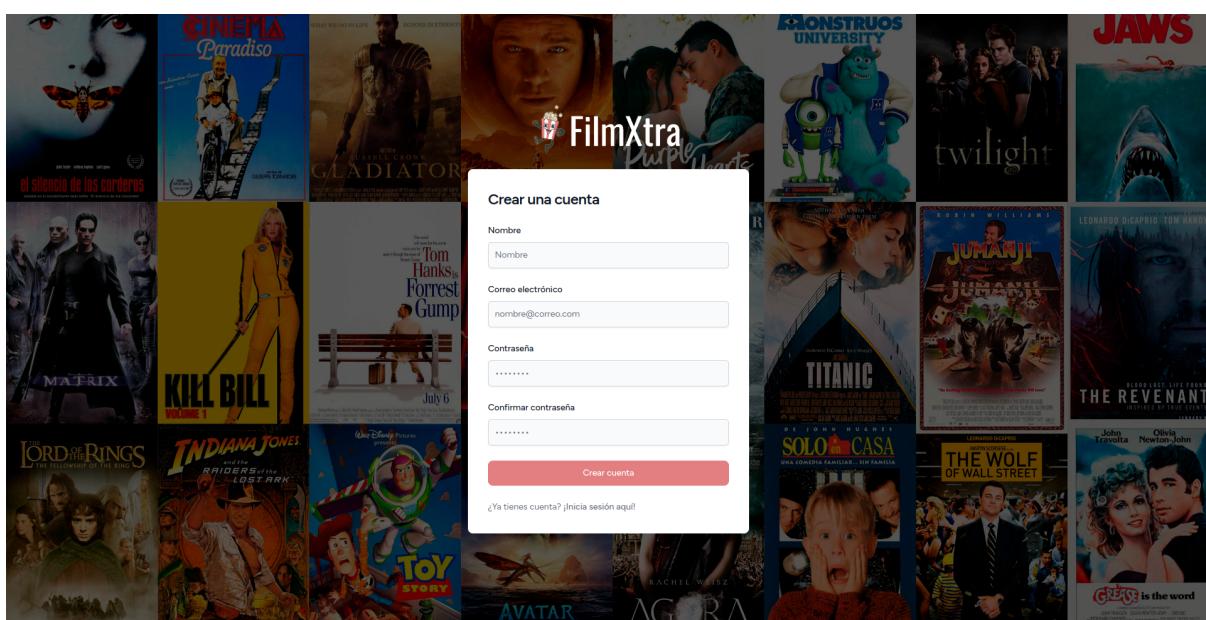
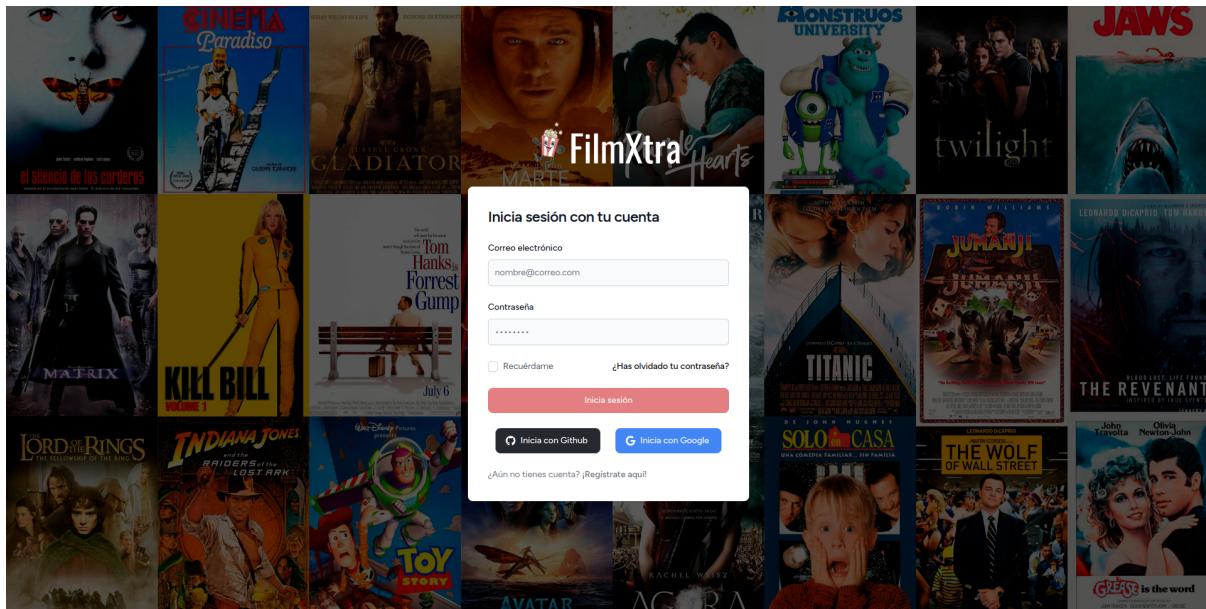
The sign-in form features a dark background with light-colored text and input fields. It includes fields for 'Your email' (with placeholder 'name@company.com') and 'Password' (with placeholder '\*\*\*\*\*'). There are checkboxes for 'Remember me' and 'Forgot password?'. A large blue 'Sign in' button is at the bottom.



The registration form also has a dark background. It includes fields for 'Your email' (placeholder 'name@company.com'), 'Password' (placeholder '\*\*\*\*\*'), and 'Confirm password' (placeholder '\*\*\*\*\*'). There is a checkbox for 'I accept the Terms and Conditions'. A large blue 'Create an account' button is at the bottom. Below the button, there is a link 'Already have an account? Login here'.

El resultado de su trabajo son dos pantallas cuidadosamente diseñadas que reflejan la estética y la funcionalidad que buscamos. Estas páginas permiten a los usuarios acceder a sus cuentas existentes o crear nuevas cuentas de manera intuitiva y segura.

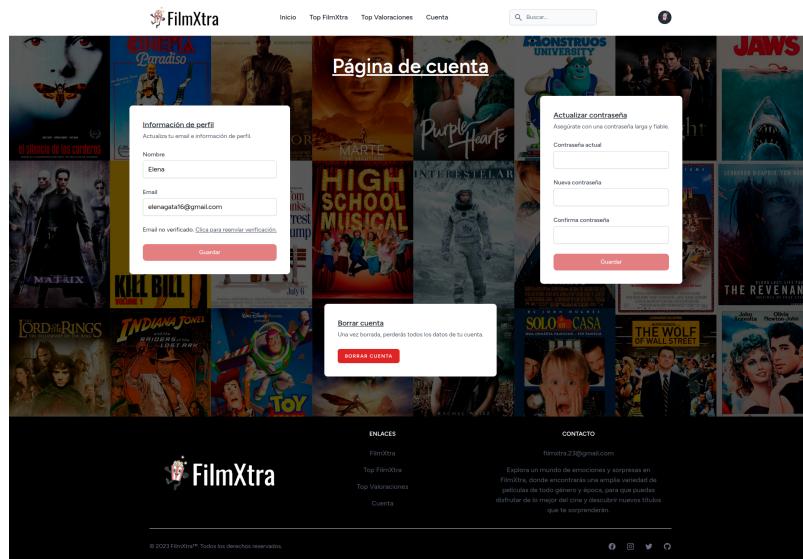
En la página de "Login", los usuarios pueden ingresar sus credenciales de cuenta para acceder a su perfil y disfrutar de todas las funciones de FilmXtra. Mientras tanto, en la página de "Register", se presenta un formulario donde los nuevos usuarios pueden proporcionar la información necesaria para crear una cuenta y unirse a nuestra comunidad.



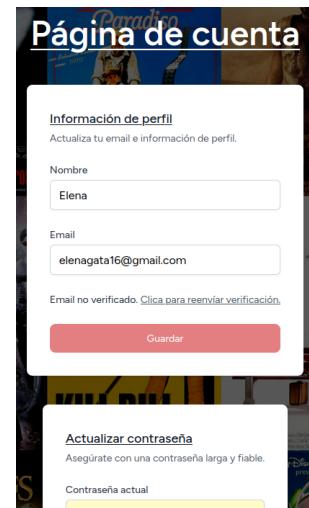
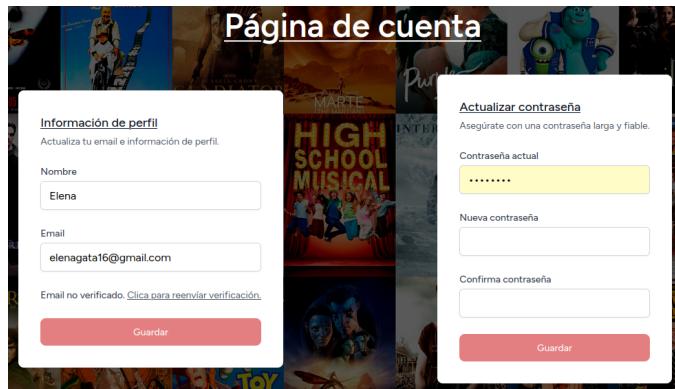
Elena se aseguró de que ambas páginas fueran atractivas visualmente, fáciles de usar y estuvieran en línea con la experiencia general de usuario que queremos ofrecer en FilmXtra.

### 3.7. Página “Edit”

Para la página de edición de cuenta de usuario, que hemos denominado "Edit" en nuestro proyecto, solo fue necesario aplicar estilos a los elementos predeterminados proporcionados por Laravel. Para lograrlo, utilicé dos divs haciendo uso de la tecnología grid proporcionada por Tailwind.

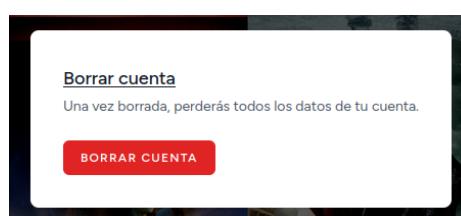


En el primer div, que consta de dos secciones, en pantallas de escritorio se divide en dos columnas, mientras que en pantallas más pequeñas se muestra en una sola columna, con cada sección ubicada una debajo de la otra. Esto permite una visualización clara y ordenada del contenido sin importar el tamaño de la pantalla.



En el segundo div, simplemente se incluyó una única columna para todas las pantallas, ya que se trata de una caja de contenido única y no requiere de divisiones adicionales. Esto garantiza la coherencia y consistencia visual en diferentes tamaños de pantalla.

Por último, un tercer div diseñado específicamente para contener el botón de borrado de cuenta. Este div se ha creado con el propósito de ofrecer a los usuarios la opción de eliminar su cuenta de manera sencilla y directa.

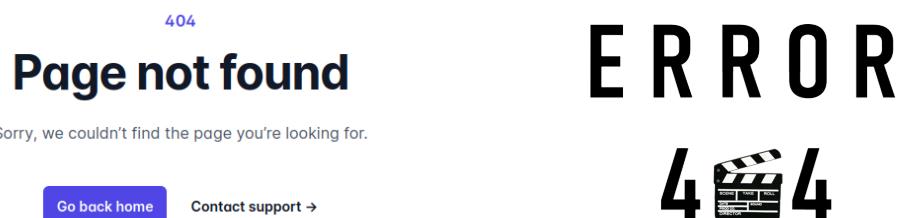


### 3.8. Página “Error 404”

También creamos una página de error para cuando se introduce una ruta que no existe. Siguiendo la sugerencia de nuestro profesor Agustín, de las asignaturas de "Diseño de interfaces web" y "Desarrollo web en entorno cliente", decidimos hacerla divertida pero sin comprometer la seriedad y el aspecto general de la web.



Para lograr esto, nos inspiramos en la página de [error 404](#) de Tailwind y la adaptamos para que encajara perfectamente con el estilo de nuestra web. La página de error consta de dos GIFs, uno como fondo de pantalla y otro ubicado en la sección principal. En esta sección, hemos utilizado una imagen personalizada, creada por Elena, una frase divertida, el logotipo de FilmXtra y un botón, creando un conjunto animado pero a la vez minimalista y serio.



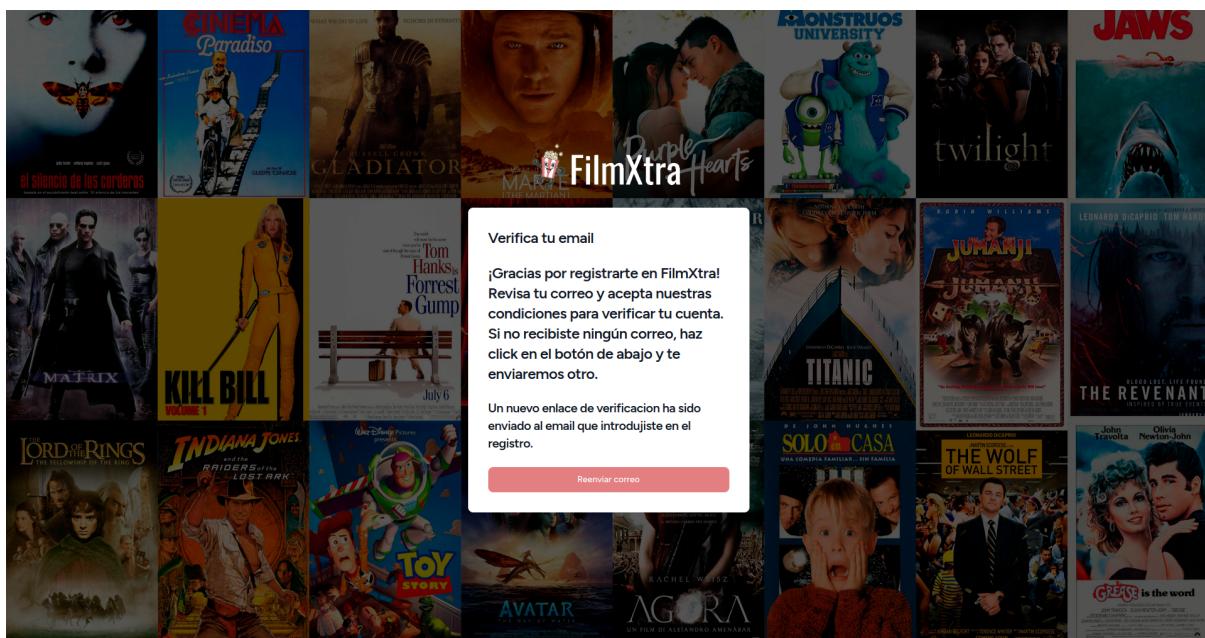
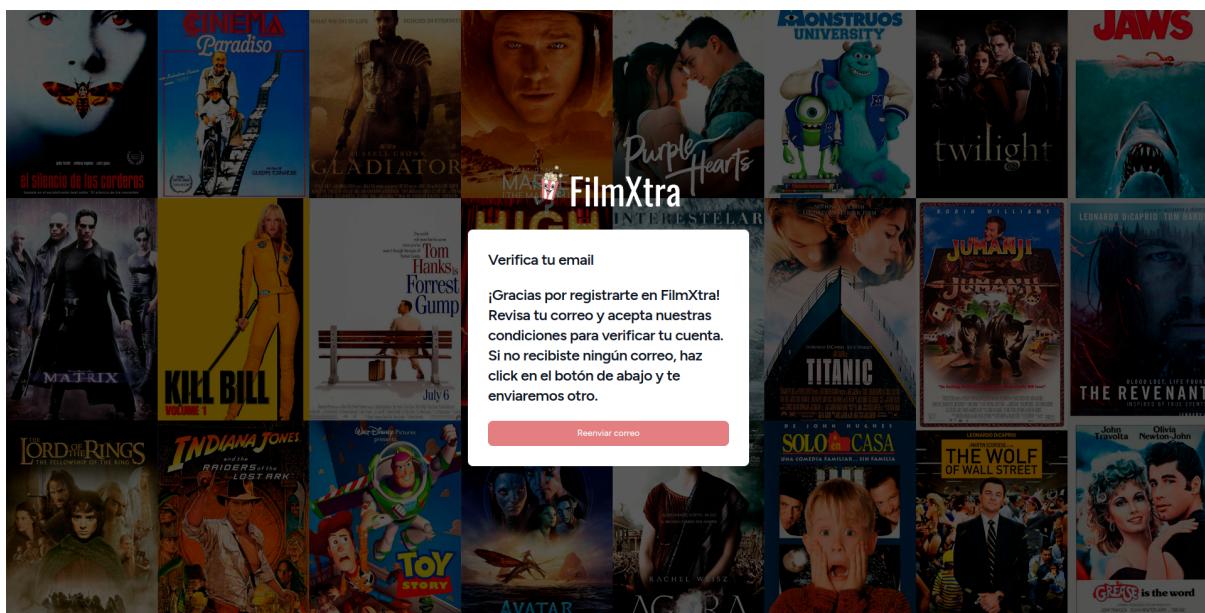
Nuestra intención al diseñar esta página de error es mantener la coherencia estética de la web y brindar una experiencia agradable incluso en situaciones de error. Consideramos que agregar un toque de humor en momentos inesperados puede ayudar a los usuarios a mantener una actitud positiva mientras navegan por nuestra plataforma.

Gracias a la adaptación y personalización de la página de error 404, hemos logrado incorporarla de manera armoniosa en todo el conjunto de nuestra web, asegurándonos de que sea coherente con la identidad visual y el tono general del sitio.

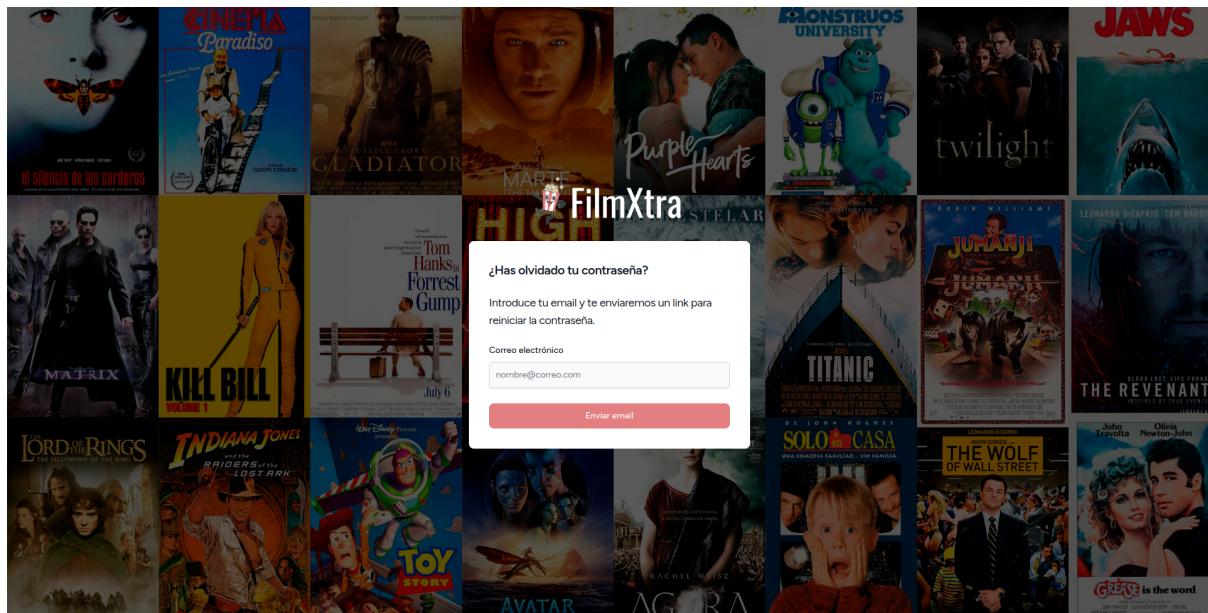
### 3.9. Páginas de verificación y restauración de contraseñas

Por último, tenemos las dos páginas restantes: la página de verificación de correo electrónico y la página de restablecimiento de contraseña. Estas páginas surgieron sobre la marcha, ya que inicialmente no se tuvieron en cuenta. Fue una situación poco común durante el proceso de creación de las páginas, pero en este caso, tanto la página de error 404 como estas dos páginas adicionales fueron creadas por solicitud de Víctor y Rodrigo.

La página de verificación de correo electrónico la encargó Rodrigo, ya que, como parte de una de sus tareas, era necesario informar a los usuarios que, además de registrarse, debían verificar su cuenta a través del correo electrónico proporcionado durante el proceso de registro. En cuanto al diseño, se optó por una apariencia simple y minimalista, en línea con el estilo general de la web. Se incluyó una frase explicativa y un botón que, al hacer clic, enviaba nuevamente el correo de confirmación.



Por otro lado, Víctor y Rodrigo encargaron la página de restablecimiento de contraseña. Al igual que la página anterior, era necesario enviar un correo electrónico para permitir a los usuarios restablecer su contraseña. Por lo tanto, se diseñó una pantalla donde los usuarios podían ingresar su dirección de correo electrónico y obtener las instrucciones necesarias para realizar el restablecimiento de contraseña.



Ambas páginas fueron desarrolladas para asegurar una experiencia fluida y segura para los usuarios, proporcionándoles la información y las herramientas necesarias para interactuar correctamente con la plataforma. A pesar de ser añadidos posteriores en el proceso de creación, se les dio importancia y se adaptaron al diseño simple y minimalista que caracteriza al resto de la web.

## 4. Adecuación del entorno de trabajo

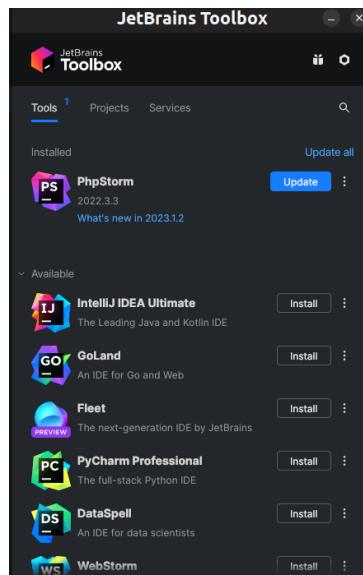
### 4.1. Instalación y configuración del IDE

Para el entorno de trabajo decidimos utilizar PhpStorm debido a que en diferentes fuentes lo consideraban como el entorno de desarrollo integrado actual más potente, para el trabajo con PHP y su framework Laravel. La decisión final vino al comprobar que para nosotros como estudiantes, se ofrecía la versión completa con una licencia gratuita por un año.

El proceso fue sencillo:

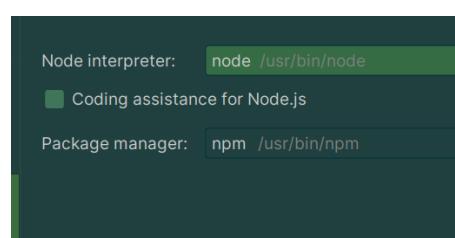
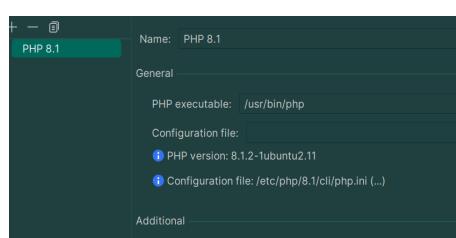
- Instalar JetBrains Toolbox (gestor de software de la propia compañía para instalar PhpStorm y otros productos, además de gestionar actualizaciones, etcétera)
- Instalar PhpStorm.
- Crear una cuenta de estudiante en Jetbrains (solo requiere el email de la institución educativa como prueba y te mandan la activación de la licencia).
- Y configurarlo para ponernos a trabajar.

Aquí podemos ver el gestor JetBrains Toolbox y como de hecho nos está ofreciendo actualizar PhpStorm:

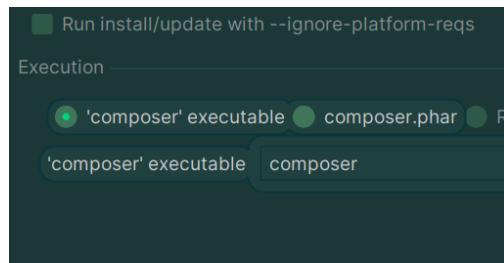


La configuración de PhpStorm es relativamente sencilla ya que gran parte de todo ello te lo dan masticado. Configuramos las rutas de algunos ejecutables (PHP, Composer, Npm):

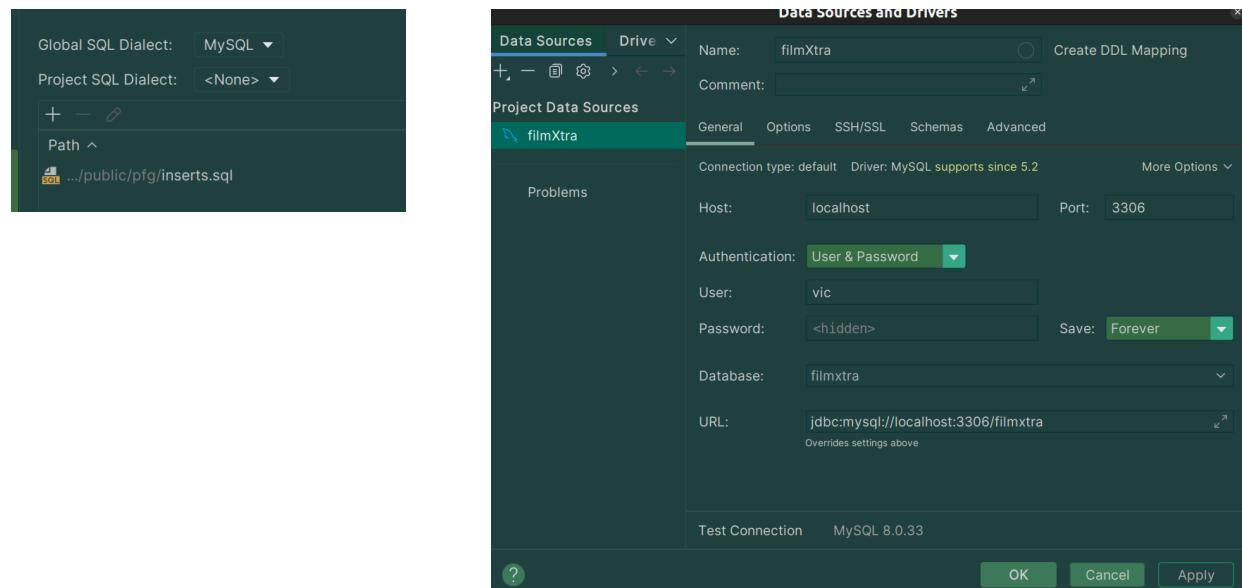
Node (NPM) y PHP:



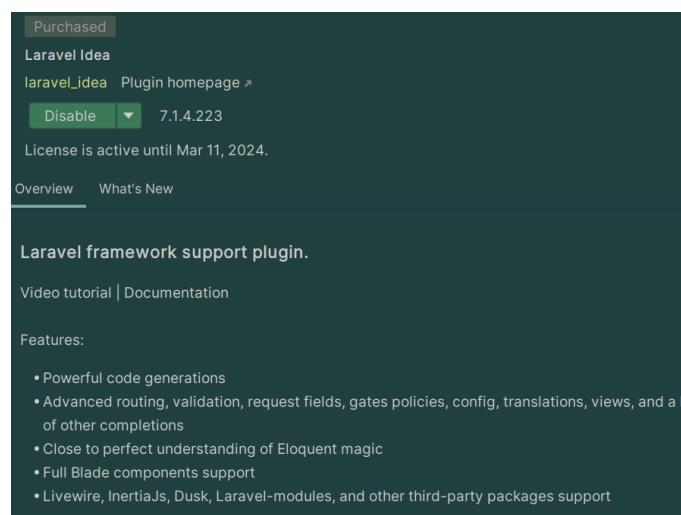
Configuración para el ejecutable de Composer (gestor de paquetes de Laravel).



Establecimiento del dialecto de MySQL y de una fuente de datos (para el acceso a MySQL desde el propio IDE) con un montón de funcionalidades, casi tantas como el servidor:



Instalación de algún complemento como Laravel Idea que contribuye a la detección de código relacionado, autocompletado y mejora de la compatibilidad con Inertia:



## Vista general del IDE:

The screenshot shows the PhpStorm IDE interface for the 'filmXtra' project. On the left, the project structure is visible with files like 'FichaValoracion.vue', 'ObtenerObraController.php', and various controller and model files. The central area has two tabs open: 'ObtenerObraController.php' and 'console'. The 'console' tab contains MySQL queries related to the database schema, such as selecting from 'generos', 'obras', and 'likes' tables. Below the code editor is a 'Database' panel showing the structure of the 'filmxtra' database with tables like 'actor', 'criticas', 'director', 'evaluaciones', and 'likes'. At the bottom, there's a results table for a query that lists countries and their counts.

pais	COUNT(*)
Alemania	2
Australia	3
España	1
Estados Unidos de América	12
Francia	2
Italia	4
Japón	1
Nueva Zelanda	3
Reino Unido	12

Podemos ver:

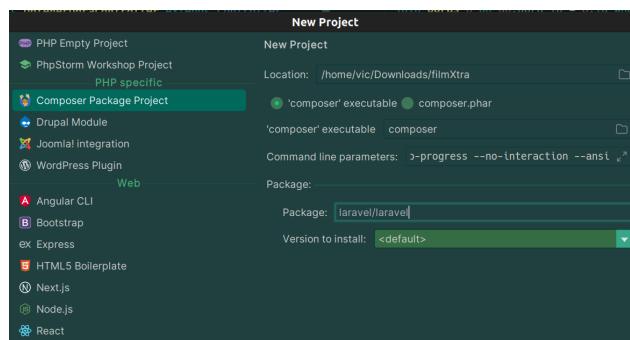
- En el panel de la izquierda está el típico navegador de archivos/directorios de todo el proyecto.
- En el centro dos ficheros abiertos: a la izquierda un controlador y a la derecha un fichero con consultas de trabajo para la base de datos.
- A la derecha tenemos el panel de acceso a la fuente de datos en el que podemos ver algunas de las tablas y otra información relacionada.
- En el panel inferior tenemos el resultado de una de las consultas pero como en otros entornos de desarrollo podemos tener la consola y muchas más posibilidades.

## 4.2. Andamiaje del proyecto

El andamiaje del proyecto es sencillo de generar, con el asistente seleccionamos: New Project -> Composer Package Project -> Y configuramos las opciones:

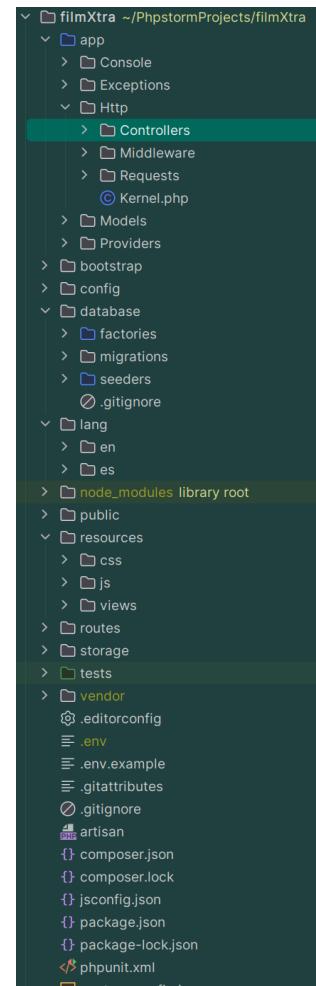
- Nombre del proyecto.
- Composer Ejecutable ya que lo tenemos instalado en el sistema.
- Versión.
- Package laravel/laravel.

Al aceptar se genera automáticamente dicho andamiaje. Veamos una captura del asistente:



A partir de ahí, como suele ser habitual con este tipo de entornos de desarrollo, se nos da una estructura general del proyecto preconfigurada que aunque es personalizable, no es muy recomendable modificarla, no solo por la complejidad que eso puede llevar en algunos aspectos sino también porque digamos que es algo que ya ha sido pensado para nosotros con programadores con más experiencia y está optimizado. Veamos un esquema general de dicha estructura:

- **/app**: uno de los directorios más importantes con subdirectorios de la índole de:
  - **/Http**: que incluye el directorio con los controladores, el middleware, la validación de las peticiones, etcétera,
  - **/Models**: los modelos de nuestra aplicación.
- **/config**: diferentes e importantes ficheros de configuración para servicios, session, vistas, etcétera.
- **/database**: engloba directorios tan importantes como las migraciones (en Laravel son las encargadas de crear las tablas de la base de datos).
- **/lang**: contiene ficheros con textos para la traducción automática a diferentes idiomas de las vistas y su contenido.
- **/public**: el directorio de acceso/raíz de la aplicación en el cual también se almacenan los materiales con los que trabaja la aplicación como las imágenes, etcétera.
- **/resources**: contiene todos los ficheros de vistas, layouts, componentes de Vue, etcétera.
- **Otros elementos**: digno de mención es el fichero '.env' cuya funcionalidad es albergar variables de configuración globales a las que la aplicación va a tener que recurrir en otros ficheros de configuración más específicos o para hacer funcionar el envío de correos electrónicos, el acceso a la capa de persistencia... Veamos una captura:



```

1 APP_NAME=filmXtra
2 APP_ENV=local
3 APP_KEY=base64:L6lH4K8woo4U3wg3KJUQuogzfGPawyRa3Plsyey5hAWg=
4 APP_DEBUG=true
5 APP_URL=http://localhost
6
7 LOG_CHANNEL=stack
8 LOG_DEPRECATIONS_CHANNEL=null
9 LOG_LEVEL=debug
10
11 DB_CONNECTION=mysql
12 DB_HOST=127.0.0.1
13 DB_PORT=3306
14 DB_DATABASE=filmxtra
15 DB_USERNAME=vic
16 DB_PASSWORD=10medieval
17
18 SERVER_PORT=80
19 SERVER_HOST=www.filmxtra.es

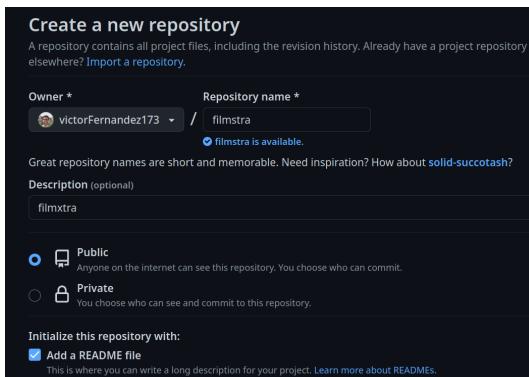
```

### 4.3. Control de versiones

La elección para el control de versiones de Git fue algo natural, dada su amplia aceptación, adopción y reconocimiento público. El proceso de configuración del repositorio es rápido y limpio.

Al haber creado el andamiaje del proyecto se aplica automáticamente una configuración de Git a nuestro proyecto que incluye además de los ficheros de configuración para el control de versiones de Git y su funcionamiento, diferentes ficheros '.gitignore' repartidos por todo el proyecto para facilitarnos y ahorrarnos este proceso. Obviamente esto es personalizable.

A continuación nos vamos a la web de GitHub y creamos un nuevo repositorio:

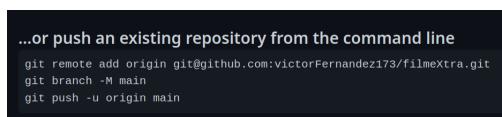


Será la propia web de GitHub la que nos da explicaciones de como enlazar dicho repositorio remoto a tu local:

- Primero eligiendo qué tipo de validación vas a elegir (HTTPS o SSH):



- Y después enlazando como digo remoto con local:



(Renombrando ya de paso la rama 'master' a 'main' parece ser que por motivos semántico-culturales.)

Para que ese tipo de enlazado funcione con validación SSH, hay que haber creado primero una clave pública y configurarla en tu cuenta de GitHub, pero es un proceso rápido y a la larga muy cómodo porque a la hora de hacer subidas o descargas no se requiere introducir contraseñas ni nada ya que se aplica el uso de tu clave pública SSH. Git ofrece tutoriales de como crear dicha clave y configurarla en nuestra cuenta:

**Generating a new SSH key**

You can generate a new SSH key on your local machine. After you generate the key, you can add the public key to your account on GitHub.com to enable authentication for Git operations over SSH.

Note: GitHub improved security by dropping older, insecure key types on March 15, 2022. As of that date, DSA keys (`ssh-dss`) are no longer supported. You cannot add new DSA keys to your personal account on GitHub.com.

RSA keys (`ssh-rsa`) with a `valid_after` before November 2, 2021 may continue to use any signature algorithm. RSA keys generated after that date must use a SHA-2 signature algorithm. Some older clients may need to be upgraded in order to use SHA-2 signatures.

- Open Terminal.
- Paste the text below, substituting in your GitHub email address.

```
$ ssh-keygen -t ed25519 -C "your_email@example.com"
```

**Adding a new SSH key to your account**

After adding a new SSH authentication key to your account on GitHub.com, you can reconfigure any local repositories to use SSH. For more information, see "Managing remote repositories".

Note: GitHub improved security by dropping older, insecure key types on March 15, 2022. As of that date, DSA keys (`ssh-dss`) are no longer supported. You cannot add new DSA keys to your personal account on GitHub.com.

RSA keys (`ssh-rsa`) with a `valid_after` before November 2, 2021 may continue to use any signature algorithm. RSA keys generated after that date must use a SHA-2 signature algorithm. Some older clients may need to be upgraded in order to use SHA-2 signatures.

Copy the SSH public key to your clipboard.

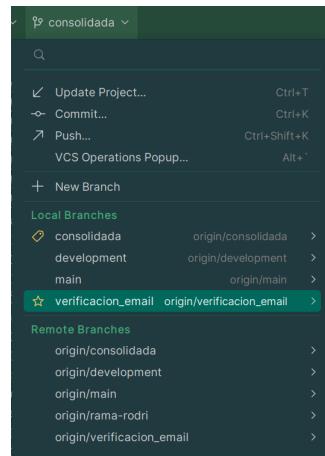
If your SSH public key file has a different name than the example code, modify the filename to match your current setup. When copying your key, don't add any newlines or whitespace.

```
$ cat ~/.ssh/id_ed25519.pub
# Then select and copy the contents of the id_ed25519.pub file
# displayed in the terminal to your clipboard
```

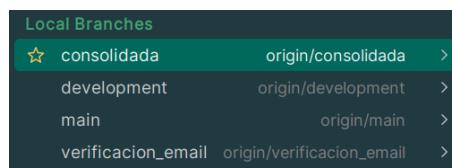
Tip: Alternatively, you can locate the hidden `.ssh` folder, open the file in your favorite text editor, and copy it to your clipboard.

Con esto ya teníamos nuestro control de versiones de manera básica, configurada. Como he mencionado, PhpStorm integra dicho control de versiones y permite muchas opciones de configuración además facilitar el manejo del control de versiones con funcionalidad de interfaz gráfica. Esto ayuda en situaciones como cuando vas a hacer un ‘commit’ y quieres ver gráficamente los cambios entre lo que vas a añadir y lo que tenías antes de haber generado cambios, o para decidir cuando hay conflictos y cambios inesperados con qué te quedas y qué deshechas.

El botón de menú de control de versiones (nos indica la rama de trabajo actual) nos permite un rápido acceso tanto a ramas locales como remotas además de otras opciones:



Además indica con qué ramas remotas están sincronizadas las locales:



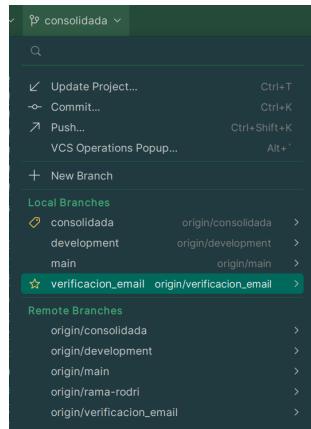
Debajo podemos ver la vista de un ‘commit’ comparando la versión previa y la actual y en el cual vamos a hacer un ‘rollback’ de uno de los ficheros porque queremos recuperar su estado previo:

The screenshot shows the 'Changes' tab in the Changelist interface. It displays a diff between two versions of a file named 'FichaValoracion.js'. The 'Before Commit' section shows the previous state of the code, and the 'After Commit' section shows the current state after a modification. A 'Rollback...' button is visible in the commit toolbar. The bottom part of the screenshot shows the detailed diff view with line numbers and changes.

```

    const existeLaEvaluacionVarComputed = computed(() => {
      if (compruebaSiExisteLaEvaluacion(page.props.auth.user['id']), page.props
        .obra[0]['id']) != existeLaEvaluacionBandera.value) {
        existeLaEvaluacionBandera.value = true;
        return 'Evaluación exitosa';
      }
      return 'Evaluación modificada exitosamente';
    });
  
```

La metodología de trabajo respecto al control de versiones tuvo cierta organización, creamos una serie de ramas en las que trabajamos cada uno de los componentes unilateralmente en local o en las que aplicamos específicamente diversas funcionalidades como puede ser Rodrigo para el tema de la validación de correo electrónico. Pero siempre sin aplicar los cambios que íbamos desarrollando a la rama main hasta que ya los habíamos consolidado. Podemos ver una captura con algunas de esas ramas en mi entorno de trabajo:



Además se aplicó una regla de seguridad a la rama 'main' que controla diferentes aspectos como que no se pudiera 'mergear' contenido a esta rama de manera automática sin control ya que como digo es la que iba a ir adoptando funcionalidades consolidadas. Veamos la configuración de dicha regla:

- Por ejemplo requiere que se soliciten pull request antes de 'mergearle' contenido y que al menos haya una aprobación.
- Una medida muy interesante es que si mientras el pull request está pendiente se le hacen nuevos 'commit', dicho 'pull request' queda anulado.
- Se chequea que las ramas que se están intentando unir a main estén actualizadas con el contenido de main.
- Otras reglas que no se ven es que esto se aplica al administrador también.

Todo siempre para asegurar que la rama main contuviera código lo más sólido posible.

The screenshot shows the 'Branch protection rule' configuration for the 'main' branch. It includes sections for 'Branch name pattern' (set to 'main'), 'Protect matching branches' (with several checkboxes like 'Require a pull request before merging', 'Require approvals', 'Require status checks to pass before merging', and 'Require branches to be up to date before merging' all checked), and a note about 'Dismiss stale pull request approvals when new commits are pushed'.

## 5. Arquitectura del proyecto

### 5.1. Modelos

El nuestro es un proyecto basado en el sistema MVC (modelo-vista-controlador) que tantas veces hemos trabajado en clase. Como tal, una vez que ya disponíamos de una base de datos sólida y el andamiaje del proyecto comenzamos el desarrollo de los modelos en base a las convenciones de Laravel.

Para la creación de los ficheros de dichos modelos nos ayudamos de un comando de Laravel que de una vez te genera el propio modelo, una migración (para crear la tabla) y un controlador para dicho modelo:

```
'php artisan make:model NombreDelModelo -mrc'
```

Laravel dispone de muchos comandos de esta índole para facilitar y acelerar la labor del desarrollador que pueden llegar a ser muy cómodos. Una de las convenciones que adoptamos a la hora de crear los modelos fue que el nombre del modelo ha de ser en singular.

De acuerdo a todas las tablas de las que disponíamos creamos sus respectivos modelos en base a la documentación recomendada por Laravel. Aclarar que esto supuso un pequeño proceso de aprendizaje bastante acelerado por cierto, pero necesario ya que como cada framework, este tiene sus particularidades. Vamos a analizar la composición de los modelos de Laravel:

### Encabezado:

- Al inicio nos encontramos con el espacio de nombre que indica donde encontrar esta clase y ayuda diferenciarla de otras que se puedan llamar igual.
- Despues tenemos las típicas importaciones que en Laravel se indican con el verbo ‘use’.
- Por último tenemos un bloque de documentación que sirve para indicar tanto al sistema como a los desarrolladores el tipo de variables de los atributos y métodos de que dispone el modelo.



```
<?php

namespace App\Models;

use Carbon\Carbon;
use Illuminate\Database\Eloquent\Collection;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

/**
 * Class Actor
 *
 * @property int $id
 * @property string $nombre
 * @property string $nombre_real
 * @property Carbon $edad
 * @property Carbon|null $defuncion
 * @property string $pais
 * @property Carbon $creado
 * @property Carbon $modificado
 *
 * @property Collection|Obra[] $obras
 */

```

## Atributos:

Esta sección es más variable y compleja:

- Por defecto Laravel sigue la convención de pluralizar el nombre del modelo para la tabla, pero ponemos otro nombre para la tabla explícitamente con 'protected \$table'. En este caso se indicó, ya que 'likes' es una tabla pivote y su nombre natural sería la combinación de las tablas que conforman la relación 'n:m' por orden alfabético ('critica\_user') y queríamos que se llamase 'likes'.
- Laravel automáticamente genera una clave primaria autoincremental sin indicar nada. En nuestro caso, para esta tabla no queríamos esa funcionalidad de ahí el uso de las variables '\$primaryKey = null' y '\$incrementing = false'.
- También asignamos a '\$timestamps' el valor 'false' evitando así que esta tabla autogenere columnas con fechas de creación y modificación.
- Aspectos muy importantes de los modelos en Laravel son el array '\$fillable' que indica qué campos van a poder ser asignables en masa o el array '\$casts' que sirve para indicar a Laravel con qué formato o tipo de dato quieres que sea recuperado esa columna cuando sea obtenida de la base de datos. También disponemos en otros modelos del array '\$hidden' que indica que campos no pueden ser asignados por formularios o en masa y así, nos aseguramos de protegerlos.

```
class Like extends Model
{
    /**
     * Tabla.
     * @var string
     */
    protected $table = 'likes';
    /**
     * primaryKey
     * @var null
     * @access protected
     */
    no usages
    protected $primaryKey = null;
    /**
     * Indica si hay auto_increment.
     * @var bool
     */
    no usages
    public $incrementing = false;
    /**
     * No timestamps
     * @var bool
     */
    no usages
    public $timestamps = false;
    /**
     * Atributos asignables.
     * @var int[]
     */
    protected $fillable = [
        'user_id',
        'critica_id',
    ];
    /**
     * Castings
     * @var string[]
     */
    protected $casts = [
        'user_id' => 'int',
        'critica_id' => 'int'
    ];
}
```

## Métodos para las relaciones

Por último veamos cómo funciona el tema de las relaciones en los modelos. A priori he de comentar que nos ha resultado bastante curioso, porque muy en la línea de Laravel sigue una sintaxis que tiende a emular el lenguaje humano y es muy descriptivo:

La estructura básica de las funciones para las relaciones consiste en que el nombre es libre pero el tipo que devuelve la función va a depender del tipo de relación: por ejemplo para una relación 1:1, el tipo que se devuelve es la clase HasOne.

Dichas relaciones en un línea 'return' devuelven la ejecución de una función de la respectiva clase (para HasOne sería la función hasOne(Parámetros...)).

```
/**
 * Obtener el trailer.
 */
no usages + victor.fernandez173
public function trailer(): HasOne
{
    return $this->hasOne(related: Trailer::class);
}

/**
 * Obtener las críticas.
 */
+ victor.fernandez173
public function criticas(): HasMany
{
    return $this->hasMany(related: Critica::class);
}

/**
 * Obtener las evaluaciones.
 */
+ victor.fernandez173
public function evaluaciones(): HasMany
{
    return $this->hasMany(related: Evaluacion::class);
}

/**
 * Obtener los géneros
 * @return BelongsToMany
 */
+ victor.fernandez173
public function generos(): BelongsToMany
{
    return $this->belongsToMany(related: Genero::class);
}
```

El parámetro que se pasará será de la clase con la que se establece la relación, por ejemplo, para obtener el Trailer de una obra se pasa como parámetro al hasOne() la clase 'Trailer::class'. Veamos un ejemplo de la captura:

- (1:1) HasOne: para este tipo de relación hay que indicar que el método 'trailer()' va a devolver la clase 'HasOne'. En el 'return' devolveremos el método con el mismo nombre y cuyo parámetro será la clase de la cual queremos obtener objetos.

## 5.2. Migraciones

Las migraciones en Laravel son ficheros que sirven para indicar cómo son las tablas que se van a crear en las bases de datos y que obviamente tienen que tener relación con los modelos y sus atributos. Básicamente constan de dos funciones 'up()' y 'down()' para crear y borrar la tabla respectivamente.

En el momento que queramos crear las tablas de las cuales ya tenemos migraciones, tan solo debemos ejecutar el comando 'php artisan migrate' y Laravel se encargará de generar dichas tablas e incluso la base de datos si no existe (dada una configuración previa en el fichero '.env' para que Laravel tenga acceso a MySQL).

Un comando importantísimo y que nos ha sido de mucha ayuda para los cambios que tuvimos que ir haciendo en la base de datos ha sido 'php artisan migrate:fresh' que básicamente lo que hace es borrar todas las tablas y volver a crearlas con los nuevos cambios que puedan contener las migraciones. Veamos un ejemplo de migración:

### Función generadora de la tabla 'up()':

- Como vemos mantiene una estructura básica: para cada columna que se quiere crear se indica el tipo de dato como un método '->string()', al cual se le pasa en forma de parámetros el nombre de la columna y algunas propiedades como pueda ser la longitud máxima, etcétera.
- En el caso de querer asignar otras propiedades a la columna solo tendríamos que encadenar métodos: '->nullable()' para indicar que puede ser null, '->constrained()' o '->unique()' para indicar restricciones o 'useCurrent()' para que los timestamps almacenen la fecha actual automáticamente.
- También se pueden indicar propiedades de la tabla como sería el motor, el charset y el collation.

```
return new class extends Migration
{
    /**
     * Migrar.
     */
    ▲ victor.fernandez173 +1
    public function up(): void
    {
        Schema::create('obras', function (Blueprint $table) {
            $table->id();
            $table->string('titulo', length: 200);
            $table->string('titulo_original', length: 200);
            $table->string('pais', length: 60);
            $table->decimal('duracion', total: 3, places: 0, unsigned: true);
            $table->string('sinopsis', length: 2500);
            $table->year('fecha');
            $table->string('productora', length: 255);

            $table->timestamp('obra::CREATED_AT')->useCurrent();
            $table->timestamp('obra::UPDATED_AT')->useCurrent();

            $table->engine = 'InnoDB';
            $table->charset = 'utf8mb4';
            $table->collation = 'utf8mb4_unicode_ci';
        });
    }

    /**
     * Anular migración.
     */
    ▲ victor.fernandez173
    public function down(): void
    {
        Schema::dropIfExists('obras');
    }
};
```

### Función destructora de la tabla:

- Para el método 'down()' en cambio no hay mucha complicación, como se puede ver, si existe la tabla que se le pasa como parámetro, la borra.

En base a la explicación del funcionamiento, podemos intuir que con la ejecución del comando 'php artisan migrate:fresh' lo que ocurre es que primero se ejecuta el método 'down()' de cada una de las migraciones y posteriormente el método 'up()' de esta manera acabamos con la base de datos renovada y con los cambios que le hubiéramos hecho a las tablas. De ahí la gran utilidad de este comando en un entorno de desarrollo como el nuestro.

### 5.3. Breeze(autenticación)

Breeze es lo que se denomina un kit de iniciación para proyectos Laravel. En concreto sirve para automáticamente se añada al andamiaje de tu aplicación elementos como las rutas, controladores y vistas correspondientes para el funcionamiento del sistema de logueo y registro de usuarios.

Breeze es un sistema minimalista que incluye una simple pero efectiva implementación de aspectos como: logueo, registro, reinicio de contraseña, modificación de perfil, borrado de cuenta e incluso verificación de email, y todo ello lo hace mediante un par de simples comandos, generando automáticamente el código y clases necesarios:

Primero hemos de instalar el paquete correspondiente:

```
composer require laravel/breeze --dev
```

Y a continuación lo incluimos en el andamiaje del proyecto con:

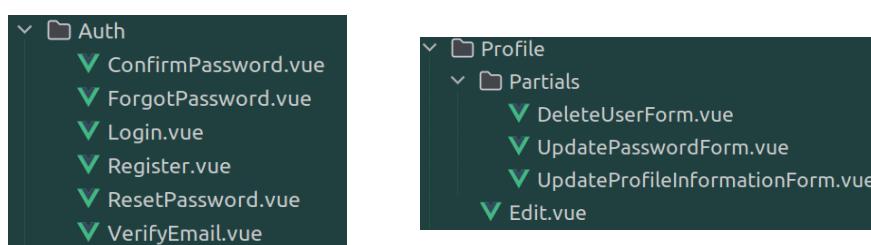
```
php artisan breeze:install vue
```

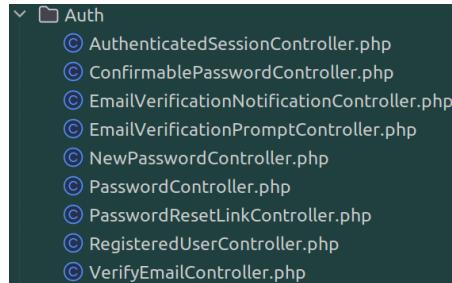
Con eso como decía, generamos: rutas en los ficheros de rutas, controladores y vistas.

Para el segundo comando podemos elegir entre generar las vistas de 'blade', sistema de vistas por defecto de Laravel o podemos decantarnos por ejemplo por una versión que genera las vistas para 'Vue'. 'Vue' va a comunicarse con Laravel mediante Inertia, una tecnología que ha sido desarrollada en colaboración de ambas partes con un objetivo: enlazar el servidor y el cliente.

Veamos unas capturas de los elementos generados por Breeze y que en algunos casos, según el uso que le hemos dado en el proyecto, hemos ido modificando de acuerdo a nuestras necesidades.

### Vistas generadas por Breeze:



**Controladores:****Rutas:**

Podemos encontrar rutas generadas en el fichero de rutas principal 'web.php'. Este contiene: 'profile.edit' para mostrar la página de perfil, 'profile.update' que sería para cuando editamos información de perfil en la vista 'UpdateProfileInformation.vue' e incluso 'profile.destroy' para eliminar una cuenta de usuario.

```
Route::middleware('auth')->group(function () {
    Route::get('/profile', [ProfileController::class, 'edit'])
        ->name('profile.edit');
    Route::patch('/profile', [ProfileController::class, 'update'])
        ->name('profile.update');
    Route::delete('/profile', [ProfileController::class, 'destroy'])
        ->name('profile.destroy');
});
```

También se genera un nuevo fichero de rutas llamado 'auth.php' que contiene más rutas relacionadas con la gestión de usuarios las cuales están agrupadas en dos bloques: para usuarios registrados y para invitados, en función de las diferentes necesidades:

- Por ejemplo, la primera ruta del bloque de invitados 'register' es un get para cargar la vista de registro de usuario mientras el 'register' (post) que la precede sería la encargada de iniciar el proceso de almacenamiento de usuarios.
- Mientras que 'login' (get) se encargará de cargar la vista de logueo, 'login' (post) almacena la sesión de un usuario.

```
// Rutas para invitados
Route::middleware('guest')->group(function () {
    Route::get('register', [RegisteredUserController::class, 'create'])
        ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);

    Route::get('login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');

    Route::post('login', [AuthenticatedSessionController::class, 'store']);

    Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
        ->name('password.request');

    Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
        ->name('password.email');

    Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
        ->name('password.reset');

    Route::post('reset-password', [NewPasswordController::class, 'store'])
        ->name('password.store');
});

// Rutas para usuarios
Route::middleware('auth')->group(function () {
    Route::get('verify-email', [EmailVerificationPromptController::class]
        ->name('verification.notice'));

    Route::get('verify-email/{id}/{hash}', [VerifyEmailController::class]
        ->middleware(['signed', 'throttle:6,1'])
        ->name('verification.verify'));

    Route::post('email/verification-notification', [EmailVerificationNotificationController::class]
        ->middleware(['store'])
        ->name('verification.send'));

    Route::get('confirm-password', [ConfirmablePasswordController::class, 'show'])
        ->name('password.confirm');

    Route::post('confirm-password', [ConfirmablePasswordController::class, 'store']);

    Route::put('password', [PasswordController::class, 'update'])
        ->name('password.update');

    Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])
        ->name('logout');
});
```

## 5.4. Rutas

Pasamos a hablar de las rutas, las cuales ya hemos mencionado en el punto anterior. Las rutas pueden considerarse la columna vertebral del backend, ya que cada solicitud que recibe el servidor se redirige a un controlador a través de una lista de rutas que asigna solicitudes a controladores o acciones.

Laravel como es habitual incorpora mucho detalle para ayudar a los desarrolladores en la creación de aplicaciones web. En nuestro caso dado que no hemos dispuesto de mucho tiempo y dada nuestra limitada experiencia no hemos creado un complejísimo sistema de rutas, pero sí que nos hemos apoyado en lo que la documentación de Laravel tenía que ofrecernos. Quizás un punto a favor habría sido mayor organización o estructuración en nuestras rutas, pero eso será sugerido como un proyecto de futuro.

Veamos una vista de todas nuestras rutas:

```
Route::get( uri: '/', [BienvenidaController::class, 'bienvenida'])->name( name: '/');

Route::post( uri: '/', [BienvenidaController::class, 'buscarTitulo'])->name( name: '/buscado');

Route::get( uri: 'obra/{titulo}', [ObtenerObraController::class, 'fichaPelcula'])->name( name: 'obra');

Route::get( uri: 'top', [TopObrasController::class, 'cargarDatos'])->name( name: 'top');

Route::get( uri: 'valoraciones', [TopValoracionesController::class, 'cargarDatos'])->name( name: 'valoraciones');

Route::get( uri: 'valoraciones/{titulo}', [FichaValoracionController::class, 'obtenerFichaValoracion'])->name
( name: 'fichaValoraciones');

Route::post( uri: '/like', [LikeController::class, 'darLike'])->name( name: 'darLike')->middleware(
 middleware: 'auth', 'verified');

Route::post( uri: 'evaluar', [EvaluacionController::class, 'evaluar'])->name( name: 'evaluar')->middleware
( middleware: 'auth', 'verified');

Route::post( uri: 'criticar', [CriticaController::class, 'criticar'])->name( name: 'criticar')->middleware
( middleware: 'auth', 'verified');
```

En base a esa captura y analizando de manera básica el código, podemos ver que la estructura esencial de una ruta en Laravel sería:

- ‘Route::get(parámetros)->(extras)’ dónde get puede ser cualquiera de los verbos HTTP.

### Parámetros:

- Primero indicaremos la url que activará la ruta. En el caso de la primera ruta, sería la ruta raíz ‘/’.
- Como segundo parámetro pasaremos un array que contendrá: el nombre del controlador que se encargará de la gestión de datos para realizar acciones y la función dentro de dicho controlador que se ejecutará para llevar a cabo dichas acciones. Para el primer ejemplo se llamará al controlador ‘BienvenidaController’ y se ejecutará su función ‘bienvenida()’.

### **Extras:**

- A esa estructura básica podemos por ejemplo, concatenarle métodos que indiquen el nombre de la ruta (`->name('/')`) para que ruta pueda ser llamada con otro nombre.
- También podremos encadenar métodos de 'middleware' para aplicar a esa ruta un control de validación de email o de autenticación como vemos en los tres últimos ejemplos de la captura.

Nuestras rutas del proyecto por nombre y (verbo):

- **'/' (get)**: para cargar la vista de bienvenida.
- **'/' (post)**: para cargar los resultados del formulario de búsqueda en la vista de Bienvenida.
- **'obra' (get)**: para cargar la ficha de información general de una película.
- **'top' (get)**: para cargar la página con el top de películas ordenadas por nota y el formulario de filtrado.
- **'valoraciones' (get)**: para cargar la página con el top de películas ordenadas por número de valoraciones.
- **'fichaValoraciones' (get)**: para cargar la ficha de valoraciones de una película.
- **'darLike' (post)** (incluye middleware para verificación de email y control de logueo): para insertar o modificar la tabla 'likes'. El middleware en concreto actúa primero comprobando si el usuario está logueado y en caso de estarlo, comprueba que el email haya sido validado, cargando en caso negativo, la vista que invita al usuario a validar su email.
- **'evaluar' (post)** (incluye middleware para verificación de email y control de logueo): para insertar o modificar la tabla 'evaluaciones'.
- **'criticar' (post)** (incluye middleware para verificación de email y control de logueo): para insertar o modificar la tabla 'criticas'.

## 5.5. Controladores

Corazón de la lógica en el sistema Modelo-Vista-Controlador. La idea es enrutar la petición HTTP hacia un controlador para que este se encargue de 'actuar' y manejar la petición.

En nuestro caso hemos creado controladores específicos para trabajar con los datos de los modelos, pero también para cargar las vistas e incluso algún controlador con lógica global para no repetir código.

Si partimos de la sección anterior sabemos que cuando se solicita una ruta, a su vez lo que se va a hacer es llamar a un controlador y en concreto a una de sus funciones, para que se encargue de hacer en nuestra aplicación web lo que le corresponde en función de esa ruta. Estos son todos los controladores que estamos manejando actualmente:

◎ BienvenidaController.php
◎ Controller.php
◎ CriticaController.php
◎ EvaluacionController.php
◎ FichaValoracionController.php
◎ InfoController.php
◎ LikeController.php
◎ ObtenerObraController.php
◎ PaginacionController.php
◎ ProfileController.php
◎ TopObrasController.php
◎ TopValoracionesController.php

Vamos a representar el proceso de funcionamiento de unos de ellos y así comprender de manera más precisa cómo se mueven los engranajes:

### Ejemplo de petición GET, ruta: '/' (Ruta de bienvenida de nuestra aplicación)

El paso inicial comienza por escribir la ruta de bienvenida de nuestro sitio web en el navegador: '[www.filmxtra.es](http://www.filmxtra.es)' de esta manera ejecutamos este bloque de las rutas:

```
Route::get( uri: '/', [BienvenidaController::class, 'bienvenida'])->name( name: '/');
```

Como podemos ver, la solicitud de dicha ruta, desencadena que se recurra al controlador 'BienvenidaController' y en concreto a su función 'bienvenida()'. Al ser una petición GET, todos los datos que se puedan pasar van a ser mediante la url y típicamente una petición de este tipo sería para simplemente cargar una vista sin mayores complicaciones, como mucho con manejo de algún parámetro. Veamos una captura del controlador:

```
/**
 * Genera un array con 16 ids al azar
 * @throws Exception
 */
2 usages  ▲ victor.fernandez173 +1
public function obtenerDoceObrasAleatorias(){
    $numPeliculas = DB::table( table: 'obras')->count();
    $peliculasId = [];
    for($i = 0; $i < 16; $i++){
        $aleatorio = random_int(1, $numPeliculas);
        while(in_array($aleatorio, $peliculasId)){
            $aleatorio = random_int(1, $numPeliculas);
        }
        $peliculasId[] = $aleatorio;
    }
    return $peliculasId;
}

/**
 * Devuelve la vista de bienvenida con esos 16 id
 * @throws Exception
 */
1 usage  ▲ victor.fernandez173
public function bienvenida(){
    return Inertia::render( component: 'Welcome', [
        'obras' => DB::table( table: 'obras')->select( columns: 'obras.titulo', 'p.ruta', 'p.alt')->join
            ( table: 'posters AS p', first: 'obras.id', operator: '=', second: 'p.obra_id')->whereIn( column: 'obras.id',
                $this->obtenerDoceObrasAleatorias())->get()
    ]);
}
```

Como decimos, el proceso continúa en el controlador ejecutando la función 'bienvenida()'. Dicha función se limita a devolver el renderizado de una vista con 'Inertia::render(vista, datos)'. Los parámetros que se le pasan son sencillos de analizar:

- **Vista:** le pasamos como un String el nombre de la vista o componente a renderizar. Laravel en su integración con Inertia, ya sabe dónde buscar las vistas ('/resources/js').
- El segundo parámetro es una consulta a la base de datos, con 'DB::table' y la particular sintaxis de Laravel. Básicamente es una consulta para obtener títulos y posters de las películas en base a doce identificadores, generados aleatoriamente

que se obtienen con la función ‘obtenerDoceObrasAleatorias()’ que se encuentra en el propio controlador como se puede ver en la captura.

Con esto acabaría el proceso del controlador que mandaría los datos a la vista y esta se renderizaría, ahora tendríamos que utilizar dichos datos en la propia vista.

## 5.6. Vistas, Flujo de información y JavaScript

Como ya se ha comentado, la comunicación entre servidor y cliente se realiza con una tecnología llamada Inertia, que se encarga concretamente de posibilitar el uso de JavaScript y la creación de ‘single page applications’ mientras seguimos usando nuestro framework de servidor, ‘uniendo los dos’.

Retomando el proceso que explicamos en el punto anterior y para ver en qué consiste dicha unión, recordaremos que mandábamos a la vista, como parte de la petición para renderizar la vista de bienvenida, un parámetro que consistía en el resultado de una consulta a la base de datos con información de doce películas seleccionadas al azar. La importancia de recordar esto se debe a que ahora que pasamos a las vistas, hemos de decir que esa información ha de manejarse en forma de ‘props’.

En Vue, los props son una suerte de parámetros que se le pasan a las vistas o componentes al igual que pasaríamos parámetros a una función. Inertia es el encargado de realizar eso, de tal manera que el bloque de datos con las películas que enviamos en el controlador y que identificabamos como ‘obras’ ahora va a estar disponible para la vista de bienvenida con ese mismo nombre y en forma de ‘prop’. Veamos qué aspecto tiene todo esto en una captura:

```

11 <script setup>
12 > import ...
13
14 onMounted( hook: () => {
15     initCarousels();
16 })
17
18 defineProps( props: [ 'obras', 'numResultados' ]);
19
20 </script>
21
22 <template>
23     <Head>
24         <title>Inicio</title>
25         <meta name="description" content="Página de bienvenida">
26     </Head>
27
28     <!-- Carrusel -->
29     <Carousel></Carousel>
30
31     <!-- Sección Principal de contenido -->
32
33     <div
34         class="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6 text-center
35             w-[75vw] m-auto my-10">
36         <div v-if="$page.props.numResultados > 0" class="col-span-full text-center
37             my-3">
38             <h4 class="text-lg">Resultados:</h4>
39         </div>
40         <!-- Posters -->
41         <Poster v-for="obra in obras" :obra="obra" :titulo=`text-lg hover:text-lg
42             sm:text-xl sm:hover:text-xl xl:text-xl xl:hover:text-xl`"
43             :info="true"/>
44     </div>
45 </template>
46

```

En la línea veintidós podemos ver cómo establecemos la función ‘defineProps()’ que básicamente lo que haría es determinar que props van a poder utilizarse. Al definir el prop ‘obras’ le estaremos diciendo a Vue que busque un objeto con datos, que tenga esa misma referencia de nombre y obviamente como sabemos que se lo estamos pasando desde el controlador, pues dará con él. A partir de aquí, ya dependerá de nosotros, lo que queramos hacer con dichos datos.

En nuestro caso como podemos ver en la línea cuarenta y tres, nos limitamos a procesar el bloque de información con un ‘v-for’, un bucle ‘for’ al uso, pero aplicado al renderizado de elementos, de tal manera que en este caso, se mostrarán tantas películas como elementos haya en el bloque de información.

## 5.7. Paginación

Debido a que en determinados momentos, nuestra aplicación puede necesitar mostrar mucha información (imágenes y otros elementos de películas), decidimos aplicar la paginación para reducir la cantidad de carga de datos y hacer más cómoda al usuario la exploración de dichos datos.

La paginación viene facilitada por Laravel como un método que se puede concatenar a una consulta a la base de datos, veámoslo en un ejemplo de un controlador:

```

30      // Consulta multicondición para filtrar películas
31      $obras = Obra::select('obras.titulo', 'p.ruta', 'p.alt',
32          DB::raw("value: AVG(e.evaluacion) AS nota_media, COUNT(*) as
33          num_valoraciones"))->join('posters AS p', 'obras.id', 'operator: =', 'p.obra_id')->leftJoin('evaluaciones AS e',
34          'obras.id', 'operator: =', 'e.obra_id')->where(
            'obras.pais', 'operator: LIKE', 'value: %' . $pais . '%')
            ->whereBetween('obras.fecha', [$d, $h])->whereHas(
                'generos', function (Builder $query) use ($genero) {
                    $query->where('genero', 'operator: like', 'value: %' . $genero . '%');
                })
            ->groupBy('obras.titulo', 'p.ruta', 'p.alt')->orderBy(
                'nota_media', 'desc')->orderBy('obras.titulo')
            ->paginate(12)->withQueryString();

```

Vemos en la última línea la función ‘->paginate(12)’ que básicamente hace que Laravel nos organice los datos para ser vistos en páginas de doce elementos en nuestro caso, mientras que ‘->withQueryString()’ le dice que pase como parámetro de la petición en la url el número de página de los datos que queremos cargar. ¿Cómo hace esto Laravel?:

The screenshot shows the MySQL Explain Plan interface with three queries listed:

- 1 DUMPS**: SELECT count(\*) AS aggregate FROM `obras`
- 3 QUERIES**:
  - 1:04:47 PM | 1.33MS | MYSQL | SELECT \* FROM `obras` limit 12 OFFSET 0
  - 1:04:47 PM | 0.15MS | MYSQL | SELECT \* FROM `posters` WHERE `posters`.`obra\_id` IN (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

A partir de nuestra consulta que genera el objeto ‘obras’, Laravel va a repartir los datos jugando con las palabras claves de ‘MySQL’ (limit y offset) de tal manera que como vemos arriba, para la primera página, obtendrá doce películas, como le habíamos pedido (limit = 12) y empezará al principio del total de todas las películas (offset = 0).

Para cada respectiva página irá jugando con ‘offset’ para entregarnos el subgrupo de datos requerido.

Además, para saber qué valor de página pasar en la consulta a la url en forma de ‘page = ’, disponemos de un componente llamado ‘Paginacion’. En base a los datos que se obtienen del controlador que como decimos, ya vienen estructurados y, mediante un ‘v-for’ se renderizan una serie en enlaces/botones.

```

7  <template>
8      <div class="mb-2" v-if="obras.links.length > 3">
9          <div class="flex flex-wrap">
10             <template v-for="(link, p) in obras.links" :key="p">
11                 <div v-if="link.url === null" class="mr-1 mb-1 px-4 py-3 leading-4
12                     text-gray-400 rounded"
13                         v-html="link.label"/>
14                 <Link v-else
15                     class="mr-1 mb-1 px-4 py-3 leading-4 rounded hover:bg-white
16                         text-base hover:border hover:border-flamingo
17                         hover:border-solid border border-white border-solid"
18                         :class="{ 'text-white bg-flamingo border-flamingo border-solid
19                             border-1 hover:text-black': link.active }" :href="link.url"
20                         v-html="link.label" preserve-state/>
21             </template>
22         </div>
23     </div>

```

Estos enlaces obtendrán el valor para su atributo ‘href’ de variables que Laravel incluye en el bloque de información estructurado para la paginación. Se incluyen variables como, el nombre del parámetro para la petición de página ‘pageName’, el valor para la primera y última página, la ruta a la que se aplica en ‘options’, etcétera. :

```

Illuminate\Pagination\LengthAwarePaginator {#1106
    #items: Illuminate\Collection {#1368 ▾
        #items: array:12 [▶]
        #escapeWhenCastingToString: false
    }
    #perPage: 12
    #currentPage: 1
    #path: "http://www.filmxtra.es/top"
    #query: []
    #fragment: null
    #pageName: "page"
    +onEachSide: 3
    #options: array:2 [▼
        "path" => "http://www.filmxtra.es/top"
        "pageName" => "page"
    ]
    #total: 83
    #lastPage: 7
}

```

## 5.8. Middleware

El uso de middleware, como mecanismo para filtrar peticiones HTTP, es aplicado en nuestra web para controlar si los visitantes son o no usuarios registrados y que puedan llevar a cabo funciones exclusivas de usuarios registrados, si lo fueran.

Laravel incluye middleware que verifica si un usuario está autenticado. Si no lo está, se redirige al usuario a la pantalla de logueo, pero si lo está se procede con la solicitud correspondiente. También hemos aplicado middleware para controlar si el email de un usuario ha sido validado o no. Habría muchas más opciones para aplicar el middleware filtrando con otros criterios las peticiones HTTP.

El middleware se aplica mediante la concatenación de funciones a las rutas en los ficheros de rutas. Veamos 3 ejemplos en nuestras rutas, que aplican el control de autenticación y de verificación de email:

```

36 Route::post('uri: /like', [LikeController::class, 'darLike'])->name('darLike')
37     ->middleware(middleware: 'auth', 'verified');
38 Route::post('uri: evaluar', [EvaluacionController::class, 'evaluar'])->name
39     ('evaluar')->middleware(middleware: 'auth', 'verified');
40 Route::post('uri: criticar', [CriticaController::class, 'criticar'])->name
41     ('criticar')->middleware(middleware: 'auth', 'verified');
```

Otra opción sería crear bloques de rutas con un middleware específico, ahorrándonos como en ese caso, repetir tres veces la aplicación del middleware, englobando esas tres rutas en dicho bloque.

## 5.9. Validación

La validación en Laravel puede realizarse principalmente de dos formas, en el propio controlador. A continuación un ejemplo de la validación en el propio controlador para la actualización de contraseña:

```

/** 
 * Actualizar el password del usuario.
 */
@victorFernandez173 +1
public function update(Request $request): RedirectResponse
{
    $validated = $request->validate([
        [
            'current_password' => ['required', 'current_password'],
            'password' => ['required', Password::defaults(), 'confirmed'],
        ],
        [
            'password.required' => 'Por favor rellene los campos', 'password.confirmed' => 'El
                password de confirmación no coincide', 'current_password.required' => '¿Y su password
                actual?', 'current_password.current_password' => 'No es el password correcto'
        ],
    ]);

    $request->user()->update([
        'password' => Hash::make($validated['password']),
    ]);

    return back();
}
```

Vemos cómo se aplica la función ‘->validate()’ a los datos del formulario obtenidos de la petición. En dicha función pasamos dos parámetros:

- El primero es un array que explicita qué campos se obtienen del formulario y que reglas se le aplican, ejemplo:
  - **Campo:** ‘current\_password’ para la contraseña actual.
  - **Regla:** ‘required’, es decir, hay que llenar este campo.
- El segundo array contendrá los mensajes que se pasarán a la vista para mostrar los correspondientes errores. Estos mensajes dependen de la combinación campo-regla:
  - **Mensaje para ‘password.required’:** pasaremos el mensaje indicado en ese elemento del array.

Aclarar que Laravel se encarga por nosotros de mandar los mensajes de error a la vista en forma de prop para que podamos manejarlos a nuestro antojo.

La otra forma de validar formularios en Laravel, sería mediante un fichero tipo ‘Request’ que sería un fichero independiente que contendrá la función ‘rules()’ para las reglas a aplicar y ‘messages()’ para los mensajes de error específicos de cada campo y regla:

```
class ProfileUpdateRequest extends FormRequest
{
    /**
     * Obtiene las reglas de validación que se aplican a la petición de actualización de
     * perfil.
     *
     * @return array<string, \Illuminate\Contracts\Validation\Rule|array|string>
     */
    no usages  ↳ victor.fernandez173 +1
    public function rules(): array
    {
        return [
            'name' => ['required', 'string', 'max:25'],
            'email' => [ 'required', 'email', 'max:60', Rule::unique( table: User::class )
                ->ignore($this->user()->id)],
        ];
    }

    ↳ victor.fernandez173 +1
    public function messages(): array
    {
        return [
            'name.required' => 'Nombre requerido',
            'name.max' => 'Máximo 25 caracteres',
            'name.string' => 'Inserte texto para el nombre',
            'email.max' => 'Máximo 60 caracteres',
            'email.required' => 'Inserte email',
            'email.email' => 'Inserte un email válido',
            'email.unique' => 'Ya existe este email'
        ];
    }
}
```

El proceso es similar pero ahora, al controlador que realiza la actualización de perfil de usuario le pasaremos como parámetro un objeto de ‘ProfileUpdateRequest’ para llevar a cabo la validación, siguiendo en esencia el mismo patrón que en la validación dentro del controlador, pero localizada en un fichero aparte:

```
→ victor.fernandez173
public function update(ProfileUpdateRequest $request):
{
```

## 5.10. Ignition y la depuración

Ignition es una ‘página de error’ para Laravel. En concreto podríamos decir que es el sistema de página de error y depuración que incluye Laravel por defecto. Hemos decidido mencionarlo con un punto específico en la memoria, ya que nos ha sido de una gran utilidad.

Ignition muestra los datos de error y depuración de manera bonita, organizada y limpia, se le puede llamar cuando queramos e incluso incluye enlaces a la documentación específica al código depurado si es detectado. Funciona igual que el que hace un ‘console.log()’ con la consola del navegador y JavaScript o un ‘System.out.println()’ en Java. Una de las opciones sería intercalar en el código de PHP un simple ‘ddd()’ (Dump, Die, Debug) es decir: se obtienen los datos que pasamos como parámetro, la ejecución muere y podemos depurar.

Vamos a llamarlo en la función ‘rules()’ para la comprobación de reglas en la validación de la actualización del perfil de usuarios. Le pasaremos la información de la petición, es decir, los datos que se mandan del formulario:

Llamada línea 18:

```

14     * @return array<string, \Illuminate\Contracts\Validation\Rule>
15     */
16     public function rules(): array
17     {
18         ddd(request());
19         return [
20             'name' => ['required', 'string', 'max:25'],
21             'email' => ['required', 'email', 'max:60', Rule::unique(User::class)->ignore($this->user()->id)],
22         ];
23     }
24

```

Formulario:

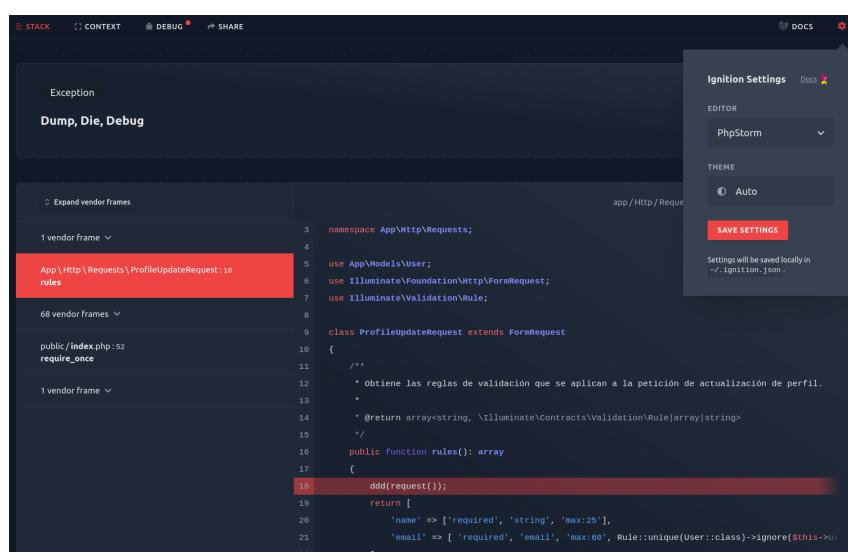
Información de perfil  
Actualiza tu email e información de perfil.

Nombre

Email

**Guardar**

Se ejecuta ‘ddd(request())’ y vemos la página de error Ignition, en ella encontraremos todo tipo de información como decimos. En la sección ‘debug’ en la barra superior de menú podremos ver que datos tiene la petición. También mostramos abierto un submenú que incluye la configuración, para indicarle cual es nuestro IDE, en la esquina superior derecha.



Datos recogidos de la petición:

(como se ve el campo name es null, ya que lo dejamos vacío en el formulario).

```

+request: Symfony... \ParameterBag {#36 ▾
#parameters: array:2 [▼
    "name" => null
    "email" => "victor.fer.gor@gmail.com"
]

```

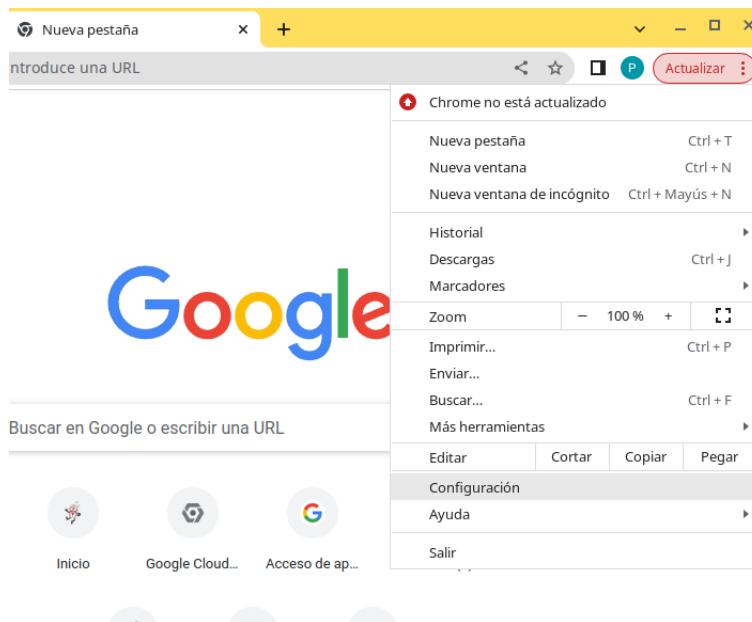
## 6. Desarrollo del proyecto

Ahora abordaremos otra fase del desarrollo del proyecto, la cual está estrechamente relacionada con las redes y ha sido liderada por Rodrigo.

### 6.1. Verificación en dos pasos

Para configurar la verificación en dos pasos, primero debemos crear una cuenta de Gmail específica para nuestra aplicación, como en nuestro caso, donde creamos una nueva cuenta llamada [filmxtra.23@gmail.com](mailto:filmxtra.23@gmail.com). Esta cuenta se utilizará para enviar correos electrónicos desde nuestra aplicación.

Una vez creada la cuenta, debemos activar la verificación en dos pasos. Para ello, accedemos a Google y hacemos clic en los tres puntos en la esquina superior derecha. Se abrirá un menú desplegable y seleccionamos "Configuración".



En la página de configuración, buscamos la opción "Gestionar tu cuenta de Google" y hacemos clic en ella. Esto nos redirigirá a 'myaccount.google.com'. A continuación, utilizamos el campo de búsqueda para buscar "verificación" y seleccionamos la opción marcada como "Verificación en dos pasos".

En este punto, es posible que se nos solicite ingresar la contraseña de la cuenta de correo. En nuestro caso, ya hemos activado la verificación en dos pasos y vinculado un teléfono, por lo que no se muestra.

Aunque pueda parecer un proceso algo confuso, es necesario realizar estos pasos. Descubrimos que es necesario crear una "contraseña de aplicación" para usarla en nuestro proyecto como la contraseña de correo. Por lo tanto, si no tienes activada la verificación en dos pasos, no podrás generar esta contraseña de aplicación.

La verificación en dos pasos está ACTIVADA desde el 5 may 2023

**DESACTIVAR**

**Segundos pasos disponibles**

Al realizar un segundo paso después de introducir la contraseña, verificamos que eres tú quien ha iniciado sesión. [Más información](#)

**Nota:** Si inicias sesión en tu cuenta de Google desde un teléfono que cumple los requisitos, se añadirán los mensajes de Google como otro método de la verificación en dos pasos.

Regresamos a la página anterior donde estábamos antes. Ahora buscamos "contraseña" y seleccionamos la opción marcada en la imagen llamada "Contraseñas de aplicaciones".

← Contraseñas de aplicaciones

Las contraseñas de aplicación te permiten iniciar sesión en tu cuenta de Google desde aplicaciones instaladas en dispositivos que no admiten la verificación en dos pasos. No tendrás que recordarlas porque solo tienes que introducirlas una vez. [Más información](#)

Nombre	Fecha de creación	Último uso
Ordenador Linux	7 may	13:42

Selecciona la aplicación y el dispositivo para los que quieras generar la contraseña de aplicación.

Seleccionar aplicación ▾ Seleccionar dispositivo ▾

**GENERAR**

Al ser redirigidos a esta página, veremos que ya hemos creado nuestra contraseña de aplicación para poder utilizarla en nuestro proyecto Laravel. Al crearla, aparecerá algo similar a lo mostrado en la imagen.

#### Contraseña de aplicación generada

Tu contraseña de aplicación para iPhone

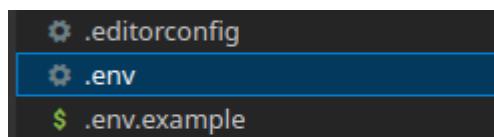
**nbrl auml tyhp glfg**

#### Cómo utilizarla

Accede a la sección de configuración de tu cuenta de Google en la aplicación o el dispositivo que estés intentando configurar. Sustituye tu contraseña por la contraseña de 16 caracteres que se muestra arriba. Al igual que la contraseña normal, esta contraseña de aplicación ofrece acceso completo a tu cuenta de Google. No tendrás que recordarla, así que no la escribas ni la compartas con nadie.

Email	securesally@gmail.com
Password	*****

A continuación, vamos a ver cómo configurar el archivo '.env' de Laravel para poder enviar correos electrónicos utilizando nuestra cuenta.



Dentro del archivo '.env', buscamos las configuraciones relacionadas con 'MAIL', que se dividen en:

- **'MAIL\_MAILER'**: hace referencia al método de transporte que se utilizará, en nuestro caso, el protocolo SMTP (Service Message Transport Protocol).
- **'MAIL\_HOST'**: indica el dominio del servidor de correo electrónico que utilizaremos. En este caso, aprovechamos el servidor público que ofrece Gmail: 'smtp.gmail.com'.
- **'MAIL\_PORT'**: aquí especificamos el puerto del servidor de correo electrónico que queremos utilizar. La principal diferencia radica en que el puerto 587 utiliza el cifrado TLS y el puerto 465 utiliza SSL. Por razones de seguridad en la entrega y recepción de correos, nos decantamos por el puerto 587 y especificamos 'TLS'.
- **'MAIL\_USERNAME'**: normalmente se ingresa la dirección de correo electrónico en este campo, pero no es estrictamente necesario.
- **'MAIL\_PASSWORD'**: este campo es importante. Aunque se denomina 'password', no debemos ingresar la contraseña de la cuenta de correo, sino la contraseña de aplicación que generamos anteriormente.
- **'MAIL\_ENCRYPTION'**: como mencionamos antes, dependiendo del puerto utilizado, especificamos aquí el tipo de cifrado correspondiente. Al utilizar el puerto 587, establecemos 'TLS'.
- **'MAIL\_FROM\_ADDRESS'**: aquí introducimos la dirección de correo electrónico desde la cual se enviarán los correos a los usuarios, en este caso, '[filmxtra.23@gmail.com](mailto:filmxtra.23@gmail.com)'.
- **'MAIL\_FROM\_NAME'**: esto viene establecido por defecto y hace referencia a una variable declarada arriba llamada 'APP\_NAME'.

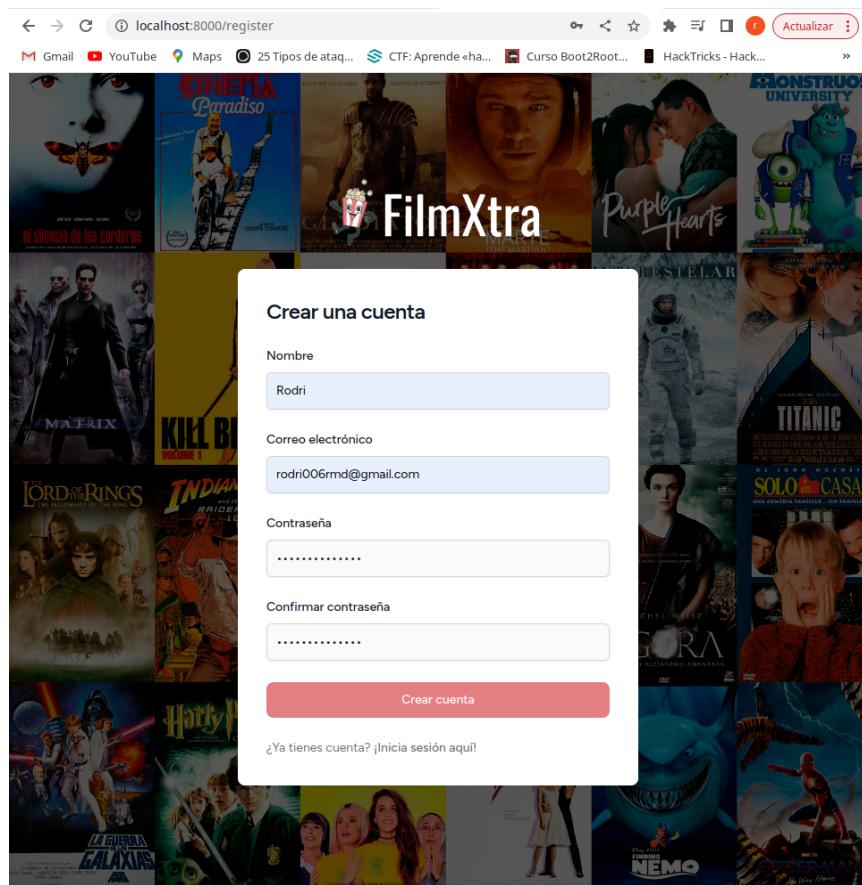
```

1 APP_NAME=Laravel
2
30
31 MAIL_MAILER=smtp
32 MAIL_HOST=smtp.gmail.com
33 MAIL_PORT=587
34 MAIL_USERNAME=filmxtra.23@gmail.com
35 MAIL_PASSWORD=ptjmanjpiiutryjc
36 MAIL_ENCRYPTION=tls
37 MAIL_FROM_ADDRESS="filmxtra.23@gmail.com"
38 MAIL_FROM_NAME="${APP_NAME}"

```

Después de configurar todo para poder enviar correos electrónicos, nos dirigimos al proyecto y su código para explicar cómo funciona la verificación en dos pasos en detalle.

Comenzamos yendo a la página de registro en nuestro proyecto. Esta página hace referencia al componente 'Register.vue' y se ve de la siguiente manera:



Una vez que hemos completado los datos, explicaremos el proceso que se seguirá cuando hagamos clic en el botón 'Crear cuenta'.

```
<form class="space-y-4 md:space-y-6" @submit.prevent="submit">
```

Como se puede observar, esto es un formulario, por lo que obviamente enviará todos los datos juntos. El evento '@submit' se refiere al envío de los datos. En este caso, se utiliza 'prevent' para evitar la funcionalidad habitual y ejecutar la variable 'submit'. Veamos qué hace esta variable.

```
const submit = () => {
  form.post(route('register'), {
    onFinish: () => form.reset('password', 'password_confirmation'),
  });
};
```

```
const form = useForm({
  name: '',
  email: '',
  password: '',
  password_confirmation: '',
  terms: false,
});
```

'Submit' es una constante que ejecuta una función cuando se la llama. Lo que hace es enviar el formulario, que es otra constante con todos los datos rellenos, a través del método POST a la ruta llamada 'register'.

Por otro lado, 'onFinish' es otra función que, en caso de que la contraseña no coincida o no cumpla con el número mínimo de caracteres, realiza un reinicio tanto en 'password' como en 'password\_confirmation'.

Ahora mostraremos la ubicación de la ruta 'register'. Para ello, navegamos a la carpeta 'routes' del proyecto y abrimos el archivo 'auth.php'. Aunque hay más rutas, nos centraremos en las que nos interesan en este momento. Comenzamos explicando que todos estos métodos GET y POST deben pasar por algo llamado 'middleware' o software intermedio.

```
// Rutas para invitados
Route::middleware('guest')->group(function () {
    Route::get('register', [RegisteredUserController::class, 'create'])
        ->name('register');

    Route::post('register', [RegisteredUserController::class, 'store']);

    Route::get('login', [AuthenticatedSessionController::class, 'create'])
        ->name('login');

    Route::post('login', [AuthenticatedSessionController::class, 'store']);

    Route::get('forgot-password', [PasswordResetLinkController::class, 'create'])
        ->name('password.request');

    Route::post('forgot-password', [PasswordResetLinkController::class, 'store'])
        ->name('password.email');

    Route::get('reset-password/{token}', [NewPasswordController::class, 'create'])
        ->name('password.reset');

    Route::post('reset-password', [NewPasswordController::class, 'store'])
        ->name('password.store');
});
```

El middleware utilizado en este caso se llama 'guest' y está declarado en el archivo 'Kernel.php' ubicado en 'app\Http'. Como se puede observar, está asociado a una clase llamada 'RedirectIfAuthenticated' que realiza lo siguiente:

```
2 references
protected $middlewareAliases = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
];
```

```
1 reference | 0 implementations
class RedirectIfAuthenticated
{
    /**
     * Handle an incoming request.
     *
     * @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next
     */
    0 references | 0 overrides
    public function handle(Request $request, Closure $next, string ...$guards): Response
    {
        $guards = empty($guards) ? [null] : $guards;

        foreach ($guards as $guard) {
            if (Auth::guard($guard)->check()) {
                return redirect(RouteServiceProvider::HOME);
            }
        }

        return $next[$request];
    }
}
```

Básicamente, verifica si el usuario ha iniciado sesión y evita que acceda a páginas para invitados. En caso de que haya iniciado sesión, se redirige a la ruta 'HOME', que explicaremos más adelante. Si no ha iniciado sesión, se pasa a ejecutar nuestro método POST en la ruta 'register' con '\$next(\$request)'.

```
Route::post('register', [RegisteredUserController::class, 'store']);
```

Ahora nos centramos en esto y ejecutamos la clase 'RegisteredUserController' en este caso, pero no se ejecuta toda la clase, especificamos que solo se ejecute la función 'store'.

```
 1 reference | 0 overrides
public function store(RegisterRequest $request): RedirectResponse
{
    $validated = $request->validated();

    $user = User::create([
        'name' => $validated['name'],
        'email' => $validated['email'],
        'password' => Hash::make($validated['password']),
    ]);

    event(new Registered($user));

    Auth::login($user);

    // Manda de vuelta a la url objetivo o sino a default
    return redirect()->intended(RouteServiceProvider::HOME);
}
```

Lo primero que se hace es crear una variable validadora que se refiere a esto.

```
 1 reference | 0 overrides
public function rules(): array
{
    return [
        'name' => ['required', 'max:25'],
        'email' => ['required', 'max:60', 'email', 'unique:' . User::class],
        'password' => ['required', 'confirmed', Password::defaults()],
    ];
}

/**
 * Mensajes de error
 * @return array
 */
1 reference | 0 overrides | prototype
public function messages(): array
{
    return [
        'name.required' => 'Introduzca nombre',
        'name.max' => 'Nombre: máximo 25 caracteres.',
        'email.required' => 'Introduzca email.',
        'email.email' => 'Formato de email no válido',
        'email.max' => 'Email: máximo 60 caracteres',
        'email.unique' => 'Email ya existente',
        'password.required' => 'Introduzca contraseña',
        'password.confirmed' => 'Confirmación fallida',
        'password.min' => 'Reglas: al menos 8 caracteres'
    ];
}
```

Esta es la clase 'RegisterRequest' que extiende de 'FormRequest' y contiene la función 'validate()' mencionada anteriormente. Sin embargo, en la función 'rules', se evalúa lo que se muestra y, en caso de que no cumpla con alguna condición, se devolverán los mensajes correspondientes en la función 'messages'.

Una vez que hemos verificado que todos los campos son válidos, se utiliza 'User::create' para crear el usuario en la base de datos.

A continuación, se realiza un detalle importante.

```
event(new Registered($user));
```

La función de eventos que estamos implementando se encarga de manejar la creación de un nuevo registro de usuario y está asociada a la clase "Registered". En el archivo "EventServiceProvider", declaramos la clase "Registered" como evento y "SendEmailVerificationNotification" como escuchador. Esto asegura que se ejecute automáticamente al momento de crear un nuevo registro de usuario.

```
1 reference
protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
];
```

Antes de profundizar en el archivo "SendEmailVerificationNotification", es importante mencionar algunos detalles relevantes en el archivo "User.php" para comprender lo siguiente. La clase "User.php" debe extender la clase "Authenticatable" (heredando funcionalidades de autenticación) y también implementar la interfaz "MustVerifyEmail" para poder utilizar las funciones relacionadas con la verificación de correo electrónico. Además, se debe utilizar la característica "Notifiable" para permitir la notificación al usuario.

```
139 references | 0 implementations
class User extends Authenticatable implements MustVerifyEmail
{
    use HasApiTokens, HasFactory, Notifiable;

class SendEmailVerificationNotification
{
    /**
     * Handle the event.
     *
     * @param \Illuminate\Auth\Events\Registered $event
     * @return void
     */
    0 references | 0 overrides
    public function handle(Registered $event)
    {
        if ($event->user instanceof MustVerifyEmail && ! $event->user->hasVerifiedEmail()) {
            $event->user->sendEmailVerificationNotification();
        }
    }
}

0 references | 0 overrides
public function hasVerifiedEmail()
{
    return ! is_null($this->email_verified_at);
}

/**
 * Mark the given user's email as verified.
 *
 * @return bool
 */
0 references | 0 overrides
public function markEmailAsVerified()
{
    return $this->forceFill([
        'email_verified_at' => $this->freshTimestamp(),
    ])->save();
}

/**
 * Send the email verification notification.
 *
 * @return void
 */
0 references | 0 overrides
public function sendEmailVerificationNotification()
{
    $this->notify(new VerifyEmail());
}

/*
 * Send the password
 */
0 references | 0 overrides
public function sendEmailPassword(){
    $this->notify(new Email());
}

/**
 * Get the email address that should be used for verification.
 *
 * @return string
 */
0 references | 0 overrides
public function getEmailForVerification()
{
    return $this->email;
}
```

Ahora, podemos continuar con la explicación del archivo "SendEmailVerificationNotification". En primer lugar, se recupera el usuario creado previamente, al cual se accede mediante \$event->user. Luego, se verifica si este usuario tiene una instancia de "MustVerifyEmail" y si su correo electrónico aún no ha sido verificado.

Si se cumplen estas condiciones, se ejecuta la función "sendEmailVerificationNotification()", que se encuentra en la interfaz "MustVerifyEmail" mencionada anteriormente. Esta función se encarga de enviar el correo de verificación.

El archivo "MustVerifyEmail" contiene varias funciones, entre las cuales se encuentra "sendEmailVerificationNotification()". Esta función utiliza "\$this" para hacer referencia a los datos del usuario y llama a "notify()", pasando como parámetro la creación de una nueva verificación de correo electrónico.

```

    public function toMail($notifiable)
    {
        $verificationUrl = $this->verificationUrl($notifiable);

        if (static::$toMailCallback) {
            return call_user_func(static::$toMailCallback, $notifiable, $verificationUrl);
        }

        return $this->buildMailMessage($verificationUrl);
    }

    /**
     * Get the verify email notification mail message for the given URL.
     *
     * @param string $url
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    1 reference | 0 overrides
    protected function buildMailMessage($url)
    {
        return (new MailMessage)
            ->subject(Lang::get('Verify Email Address'))
            ->line(Lang::get('Please click the button below to verify your email address.'))
            ->action(Lang::get('Verify Email Address'), $url)
            ->line(Lang::get('If you did not create an account, no further action is required.'));
    }

    /**
     * Get the verification URL for the given notifiable.
     *
     * @param mixed $notifiable
     * @return string
     */
    1 reference | 0 overrides
    protected function verificationUrl($notifiable)
    {
        if (static::$createUrlCallback) {
            return call_user_func(static::$createUrlCallback, $notifiable);
        }

        return URL::temporarySignedRoute(
            'verification.verify',
            Carbon::now()->addMinutes(Config::get('auth.verification.expire', 60)),
            [
                'id' => $notifiable->getKey(),
                'hash' => sha1($notifiable->getEmailForVerification()),
            ]
        );
    }
}

```

En cuanto a la clase "VerifyEmail", cuando se crea una instancia de esta clase, se ejecuta la función "toMail", que a su vez llama a "buildMailMessage" para construir el cuerpo del mensaje que se enviará por correo electrónico, y a "verificationUrl" para obtener una URL que incluye el ID del usuario, su hash y el tiempo de expiración.

Continuando con la ejecución del archivo "RegisteredUserController", luego de enviar el correo electrónico, se autentica al usuario, lo cual es importante debido a la extensión de la clase "User" como "Authenticatable". Por último, se redirige al usuario a la ruta objetivo definida en la clase "RouteServiceProvider::HOME".

```

Auth::login($user);

// Manda de vuelta a la url objetivo o sino a default
return redirect()->intended(RouteServiceProvider::HOME);

```

```

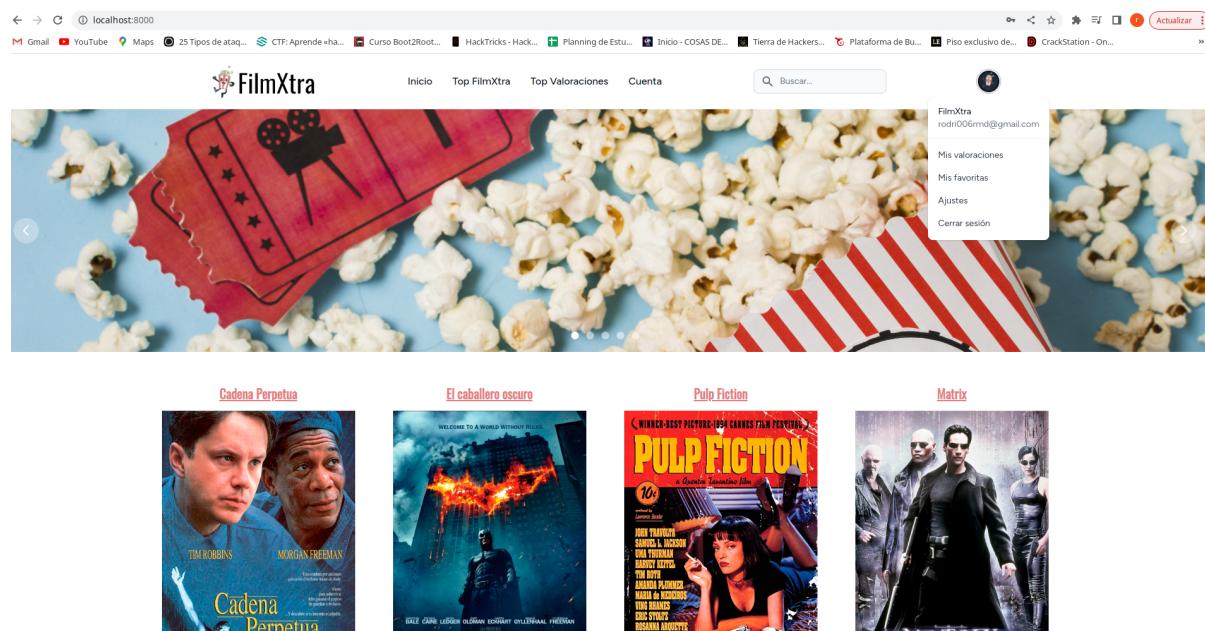
public const HOME = '/';

```

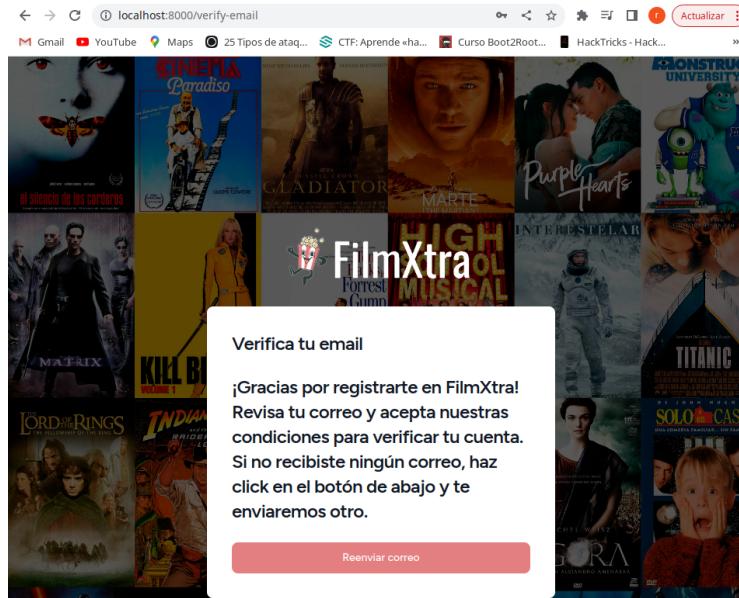
Esta ruta se encuentra en el archivo "web.php" y ejecuta la función "bienvenida" del controlador "BienvenidaController". En esta función, se renderiza el componente "Welcome" en una etiqueta Inertia, y se le pasa como "props" una consulta de MySQL que obtiene doce películas al azar, las cuales son llamadas "obras".

```
Route::get('/', [BienvenidaController::class, 'bienvenida'])->name('/');
```

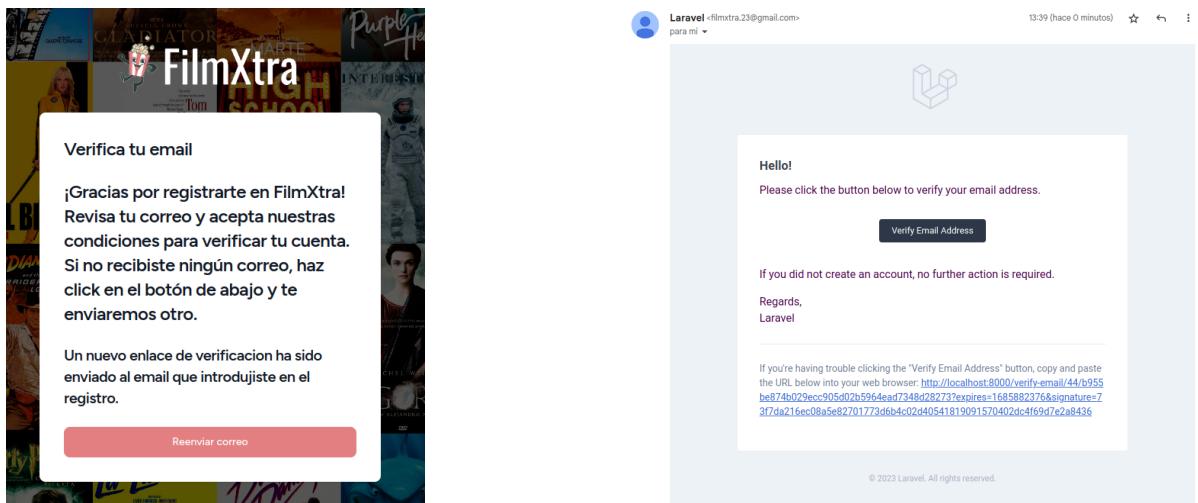
```
public function bienvenida(){
    return Inertia::render('Welcome', [
        'obras' => DB::table('obras')->select()
    ]);
}
```



Esta es la representación visual de la página resultante, donde en la esquina superior derecha se muestra que el usuario ha sido autenticado, pero no verificado. Si intentamos interactuar con otros usuarios, por ejemplo, dar "like" a un comentario, se muestra un mensaje que indica que no se puede interactuar hasta que se verifique el correo electrónico. En caso de no haber recibido el correo de verificación, se puede hacer clic en "Reenviar correo".



Cuando se hace clic en "Reenviar correo", se muestra un mensaje que confirma que se ha enviado un nuevo correo de verificación.



Como se puede ver, el correo de verificación ha llegado. Al hacer clic en él, se realiza una petición GET a `http://localhost:8000/verify-email/(ID de usuario)/(hash de usuario)`, y esta ruta se encuentra en el archivo "auth.php" que mencionamos anteriormente. En este caso, tanto las solicitudes GET como POST deben pasar por el middleware "auth", que verifica si el usuario está autenticado antes de acceder a cualquiera de estas rutas.

```
// Rutas para usuarios
Route::middleware('auth')->group(function () {
    Route::get('verify-email', EmailVerificationPromptController::class)
        ->name('verification.notice');

    Route::get('verify-email/{id}/{hash}', VerifyEmailController::class)
        ->middleware(['signed', 'throttle:6,1'])
        ->name('verification.verify');

    Route::post('email/verification-notification', [EmailVerificationNotificationController::class, 'store'])
        ->middleware('throttle:6,1')
        ->name('verification.send');

    Route::get('confirm-password', [ConfirmablePasswordController::class, 'show'])
        ->name('password.confirm');

    Route::post('confirm-password', [ConfirmablePasswordController::class, 'store']);

    Route::put('password', [PasswordController::class, 'update'])->name('password.update');

    Route::post('logout', [AuthenticatedSessionController::class, 'destroy'])
        ->name('logout');
});

Route::get('verify-email/{id}/{hash}', VerifyEmailController::class)
    ->middleware(['signed', 'throttle:6,1'])
    ->name('verification.verify');
```

Una vez que se verifica que el usuario está autenticado, nos encontramos con dos middleware propios en nuestra ruta. El middleware "signed" que verifica la firma de la URL, y el middleware "throttle" que se utiliza para controlar el tráfico, con parámetros que establecen el número máximo de solicitudes y el intervalo de tiempo en minutos.

Después de pasar por esto, se ejecuta el controlador "VerifyEmailController". Es interesante mencionar que verifica la función "authorize()" del parámetro "EmailVerificationRequest", que hace referencia a la solicitud de verificación de correo electrónico.

```

class VerifyEmailController extends Controller
{
    /**
     * Marca el email autenticado del usuario como verificado.
     */
    0 references | 0 overrides
    public function __invoke(EmailVerificationRequest $request): RedirectResponse
    {
        if ($request->user()->hasVerifiedEmail()) {
            return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');
        }

        if ($request->user()->markEmailAsVerified()) {
            event(new Verified($request->user()));
        }

        return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');
    }
}

public function authorize()
{
    if (! hash_equals((string) $this->user()->getKey(), (string) $this->route('id'))) {
        return false;
    }

    if (! hash_equals(hash1($this->user()->getEmailForVerification()), (string) $this->route('hash'))) {
        return false;
    }

    return true;
}

```

En este punto, surgieron varios problemas, como cuando se comprobaba el ID y se recibía supuestamente un valor nulo. Una vez que se verifica correctamente, el archivo "VerifyEmailController" vuelve a comprobar si el usuario ya tiene el correo electrónico verificado. En caso afirmativo, se redirige al usuario a la ruta HOME. Si el correo electrónico no ha sido verificado, se marca como verificado y se redirige al usuario a la ruta HOME concatenando "?verified=1" al final.

```

if ($request->user()->hasVerifiedEmail()) {
    return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');
}

if [$request->user()->markEmailAsVerified()]
| event(new Verified($request->user()));
}

return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');

```

	id	name	username	age	country	email	email_verified_at	password	remember_token	google_id	created_at	updated_at
46	Rodri	NULL	NULL	NULL	rodr1006rmd@gmail.com	NULL	\$2y\$10\$UHCZirkAqkVVlbCDLwk870C1Lv.D4H408inv4Z.PfjgQWEVAho8T0	NULL	NULL	2023-06-04 17:26:42	2023-06-04 17:26:42	

	id	name	username	age	country	email	email_verified_at	password	remember_token	google_id	created_at	updated_at
46	Rodri	NULL	NULL	NULL	rodr1006rmd@gmail.com	2023-06-04 17:27:17	\$2y\$10\$UHCZirkAqkVVlbCDLwk870C1Lv.D4H408inv4Z.PfjgQWEVAho8T0	NULL	NULL	2023-06-04 17:26:42	2023-06-04 17:27:17	

Aquí se presentan algunos ejemplos donde se muestra el campo "email\_verified\_at" como nulo cuando no se ha verificado y con la fecha y hora de verificación en el segundo ejemplo.

También se muestra el "?verified=1" en la URL y se demuestra que se puede dar "like" después de la verificación.

- **Críticas de nuestros usuarios:**
- Marco: Gran saga, pero esta me decepcionó un poco. (hace 2 días) - Likes: 25
- Paithoon: Ésta me gustó algo menos que las otras películas de la saga, pero en general, como todas las pelis de la saga, en su conjunto están bastante bien. recomiendo que se hagan maratón de la saga, será tiempo bien invertido XD (hace 2 días) - Likes: 25
- [...]

A continuación, vamos a comentar algunos de los errores más comunes que encontramos durante este proceso y que recordamos mejor.

En primer lugar, cabe mencionar que durante un tiempo utilizamos sesiones para jugar con la verificación en dos pasos. Sin embargo, encontramos que esto generaba problemas al introducir nuevas funcionalidades, por lo que decidimos retroceder y aprovechar las capacidades que Laravel nos proporciona. Si bien el enfoque anterior "funcionaba", había momentos en los que era posible "hacer trampas" para verificar o autenticar cuentas, lo que hacía que el sistema perdiera su propósito.

Por ejemplo, en el "RegisteredController", al inicio lo configuré de la siguiente manera:

```
public function store(RegisterRequest $request): RedirectResponse | Response
{
    $validated = $request->validated();

    $user = User::create([
        'name' => $validated['name'],
        'email' => $validated['email'],
        'password' => Hash::make($validated['password']),
    ]);

    event(new Registered($user));

    session(['user' => $user]);

    return redirect('verify-email');
}
```

Aquí se almacenaba el objeto \$user en la sesión y luego se redirigía a la página de verificación de correo electrónico.

```
public function __invoke(EmailVerificationRequest $request): RedirectResponse
{
    /*dd($request);*/
    error_log('AAAAAAAAAAAAAAAAAAAAA');
    error_log($request);
    error_log('AAAAAAAAAAAAAAAAAAAAA');
    // $request->user()
    /*dd($request);*/
    if ($request->get('user')->hasVerifiedEmail()) {
        /*dd(session('user'))*/
        error_log('EMAIL VERIFICADO VERIFY EMAIL CONTROLLER');
        Auth::login($request->get('user'));
        return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');
    }
    /*dd(session('user'))*/
    if ($request->get('user')->markEmailAsVerified()) {
        error_log('NUEVO VERIFICADO VERIFY EMAIL CONTROLLER');
        event(new Verified($request->get('user')));
        Auth::login($request->get('user'));
    }

    error_log('POR AQUÍ VA');

    return redirect()->intended(RouteServiceProvider::HOME.'?verified=1');
}
```

Otro ejemplo se encuentra en el "VerifyEmailController", donde además de realizar numerosas pruebas con comentarios y otras cosas para comprender cómo funcionaba todo, se utilizaban sesiones en las condiciones para verificar si el correo electrónico estaba verificado y para marcarlo como verificado.

También solo autenticábamos la cuenta cuando se cumplían una de las dos condiciones anteriores.

```

Error
Call to a member function getKey() on null

public/index.php:52
require_once

PHP 8.1.2-1ubuntu2.11 | © 10.7.1

```

Este error era muy común y al principio fue bastante complicado de resolver, ya que aunque el error se mostraba en ese archivo, en realidad provenía de otro lugar debido a esto:

```

public function authorize()
{
    if (! hash_equals((string) $this->user()->getKey(), (string) $this->route('id'))) {
        return false;
    }

    if (! hash_equals(sha1($this->user()->getEmailForVerification()), (string) $this->route('hash'))) {
        return false;
    }

    return true;
}

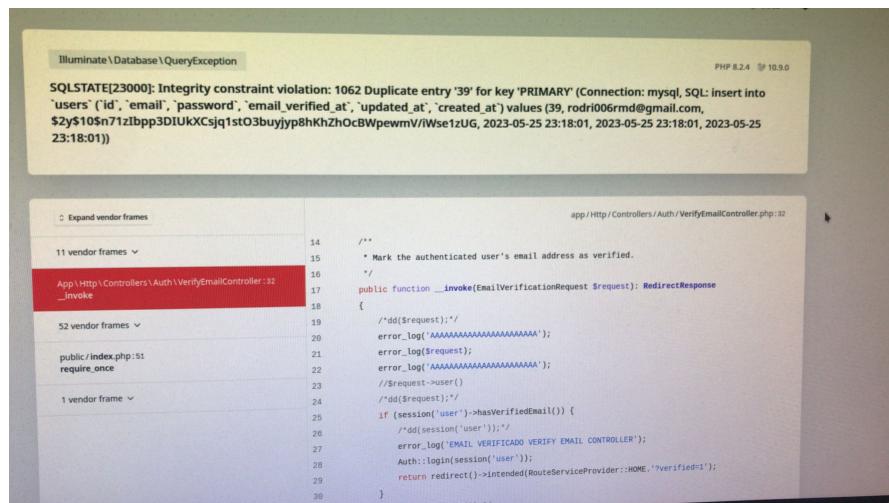
```

Aquí, cambiamos los dos "\$this->user" por "session('user')". A largo plazo, esto causaba problemas cuando se realizaban cambios en otros archivos, lo que provocaba que los valores llegaran como nulos y generaran el error mencionado anteriormente.

También encontramos el error de que cuando te registrabas y se te redirigía a la página de verificación de correo electrónico, si accedías a la ruta raíz aparecías como autenticado y verificado.

Otro problema que nos llevó más tiempo resolver, y aunque intentamos solucionarlo la última vez que surgió al modificar otros archivos, no logramos encontrar el fallo, fue el siguiente.

Para contextualizar, este problema surgía cuando hacíamos clic en el enlace de verificación de correo electrónico y nos mostraba el siguiente mensaje de error:



A simple vista, parece obvio porque nos indica que hay una entrada duplicada, y efectivamente el correo electrónico ya estaba registrado, por eso se mostraba este mensaje.

Estas dos últimas imágenes adjuntas corresponden al archivo 'Model.php', que contiene muchas funciones, incluyendo la función 'save()', que se ejecuta al intentar guardar el campo 'email\_verified\_at'.

El problema estaba en la línea donde se encontraba '\$this->exists'. Esta condición siempre se evaluaba como si el usuario no existiera, lo que llevaba al bloque 'else' y, por supuesto, al intentar insertarlo, se generaba el mensaje de error mencionado anteriormente.

```

1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156

public function save(array $options = [])
{
    $this->mergeAttributesFromCachedCasts();

    $query = $this->newModelQuery();

    // If the "saving" event returns false we'll bail out of the save and return
    // false, indicating that the save failed. This provides a chance for any
    // listeners to cancel save operations if validations fail or whatever.
    if ($this->fireModelEvent('saving') === false) {
        return false;
    }

    // If the model already exists in the database we can just update our record
    // that is already in this database using the current IDs in this "where"
    // clause to only update this model. Otherwise, we'll just insert them
    // (which is typically an auto-increment value managed by the database).
    if ($this->exists) {
        $saved = $this->isDirty() ?
            $this->performUpdate($query) : true;
    }
    else {
        $saved = $this->performInsert($query);

        if (! $this->getconnectionName() &&
            $connection = $query->getconnection() &&
            $this->setconnection($connection->getName()));
    }

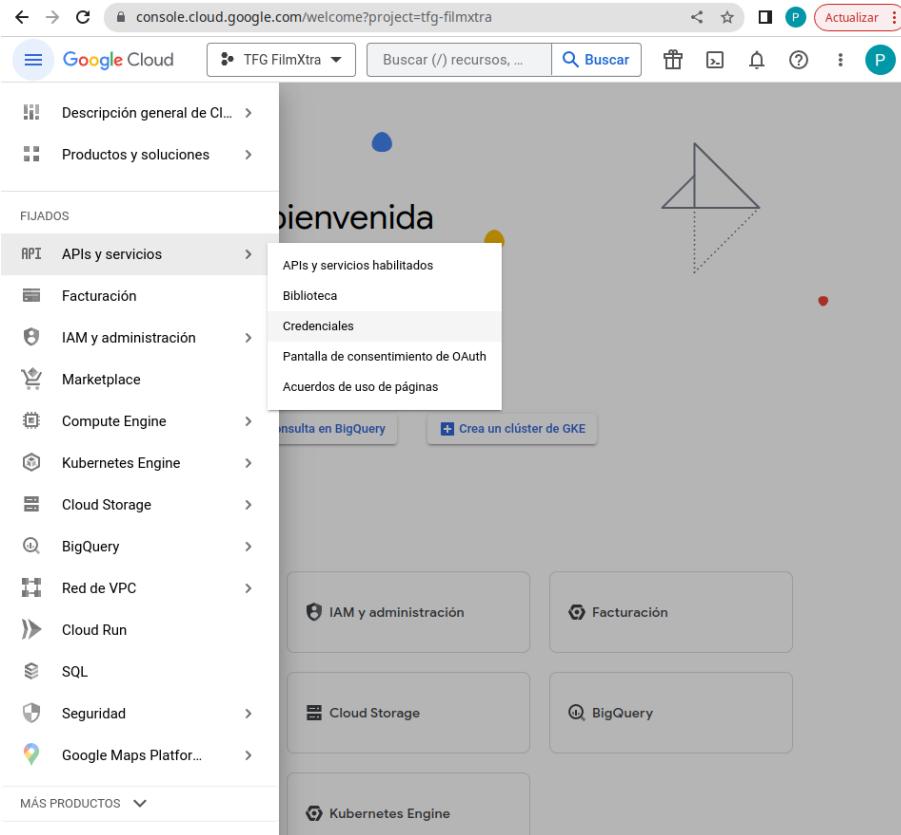
    // If the model is successfully saved, we need to do a few more things once
    // that is done. We will call the "saved" method here to run any actions
    // we need to happen after a model gets successfully saved right here.
    if ($saved) {
        $this->finishSave($options);
    }
}

return $saved;
}

```

## 6.2. Google OAuth 2

Para configurar la autenticación de Google, primero debemos dirigirnos a la página <https://console.cloud.google.com/> para crear las credenciales necesarias que utilizaremos en nuestro proyecto.



En el menú desplegable que se encuentra a la izquierda, seleccionamos "Apis y servicios" y luego hacemos clic en "Credenciales".

This screenshot shows the 'Credenciales' (Credentials) page. The left sidebar has 'Credenciales' selected. The main area has a heading 'Claves de API' with a table showing no keys available. Below it is a section for 'ID de clientes OAuth 2.0' with a table showing one client key: 'FilmXtra' (Created 31 may 2023, Type: Application, Client ID: 1001617616839-100..., web). At the bottom is a section for 'Cuentas de servicio' with a table showing no service accounts available.

Si aún no tienes un proyecto, primero necesitas crear uno nuevo. Como nosotros si lo tenemos, hacemos clic en el enlace "TFG FilmXtra" ubicado en la parte superior.

En la pestaña que se abre, verás que ya tenemos un proyecto creado. Sin embargo, si deseas crear uno nuevo, simplemente haz clic en el botón de creación ubicado en la parte superior derecha.

Una vez hayas seleccionado "TFG FilmXtra" como proyecto, serás redirigido a la página anterior con la configuración correspondiente al proyecto seleccionado.

Para generar las credenciales de OAuth, haz clic en el botón "+ CREAR CREDENCIALES" como se muestra en la interfaz. Luego, se desplegarán varias opciones y debes seleccionar "ID de cliente de OAuth".

Una vez aquí, en nuestro caso, seleccionamos "Aplicación Web" como opción para continuar.

The screenshot shows the configuration of an OAuth consent screen. It includes fields for the client name ('Nombre \*' with 'FilmXtra'), a note about authorized domains, a section for JavaScript origins ('Orígenes autorizados de JavaScript'), and a section for authorized redirect URIs ('URI de redireccionamiento autorizados'). Both sections include an 'AGREGAR URI' button and a note about configuration taking effect within 5 minutes to several hours. Buttons for 'GUARDAR' and 'CANCELAR' are at the bottom.

A continuación, se mostrarán varias opciones y es importante prestar atención al apartado de "URI de redireccionamiento autorizados", ya que deberemos proporcionar la URL a la cual Google OAuth responderá cuando necesitemos hacer uso de él.

#### Additional information

ID de cliente	1001617616839-100jdh5vf8an24fdpils2hrin4dkpeia.apps.googleusercontent.com
Fecha de creación	31 de mayo de 2023, 19:12:22 GMT+2
<b>Secretos del cliente</b>	
Secreto del cliente	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
Fecha de creación	31 de mayo de 2023, 19:12:22 GMT+2
Estado	<input checked="" type="checkbox"/> Habilitado
<a href="#">+ ADD SECRET</a>	

Después de completar todos los campos, obtendremos el "ID de cliente" y el "Secreto del cliente", los cuales utilizaremos más adelante.

Antes de continuar con la configuración en el proyecto, me gustaría resaltar un detalle que considero importante, ya que, en mi caso, me funcionó correctamente.

The screenshot shows the Google Cloud Platform API library interface. It displays the OAuth consent screen configuration under the 'Pantalla de consentimiento de OAuth' tab. The configuration includes the client ID (1001617616839-100jdh5vf8an24fdpils2hrin4dkpeia.apps.googleusercontent.com), the secret key (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), and the state (Habilitado). Below this, there are sections for 'Estado de verificación' (with a note about the verification screen not being mandatory) and 'Estado de publicación' (set to 'En producción'). The 'Tipo de usuario' is listed as 'Externo'. Navigation links for 'Acuerdos de uso de páginas' and 'Biblioteca' are also visible.

En el panel de la izquierda, seleccionaremos la opción "Pantalla de consentimiento de Oauth" en el desplegable. En el apartado "Estado de publicación", elegiremos "En producción". Esto permitirá que cualquier persona que tenga una cuenta de Google pueda acceder.

Ahora, procederemos a la configuración en nuestro proyecto. Para implementar esta autenticación, utilizaremos "Socialite", un paquete oficial de Laravel que proporciona una interfaz sencilla para autenticar usuarios a través de proveedores de autenticación social como Google, Github, Twitter, entre otros.

```
> vendor
  > .editorconfig
  > .env
  > .env.example
  > .gitattributes
```

```
60  GITHUB_CLIENT_ID=0ea22dc46749f966409f
61  GITHUB_CLIENT_SECRET=bb119cb817f4da5d467273c08b99a1a9f6bed9c7
62  GOOGLE_CLIENT_ID=1001617616839-100jdh5vf8an24fdpils2hrin4dkpeia.apps.googleusercontent.com
63  GOOGLE_CLIENT_SECRET=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
```

Nuevamente, accedemos al archivo '.env', que contiene nuestras variables de entorno, y agregamos el ID de cliente y el secreto de cliente que generamos anteriormente. Luego, continuamos con el archivo 'services.php'.

```
    ],
    'google' => [
        'client_id' => env('GOOGLE_CLIENT_ID'),
        'client_secret' => env('GOOGLE_CLIENT_SECRET'),
        'redirect' => 'http://localhost:8000/auth/google/callback',
    ]
]
```

En el archivo 'services.php', creamos una variable llamada 'google' a la que le asignamos los valores de los clientes de Google, tomados del archivo '.env', y configuramos la redirección a la ruta previamente establecida, donde Google OAuth mostrará la pantalla de verificación.

```
<!--Iniciar sesión Google-->
<a :href="route('google.login')" type="button"
    <svg class="w-4 h-4 mr-2 -ml-1" ar
        Inicia con Google
    </a>
```

Empezamos mostrando el botón de Google, el cual, al hacer clic en él, nos redirigirá a la ruta 'google.login'.

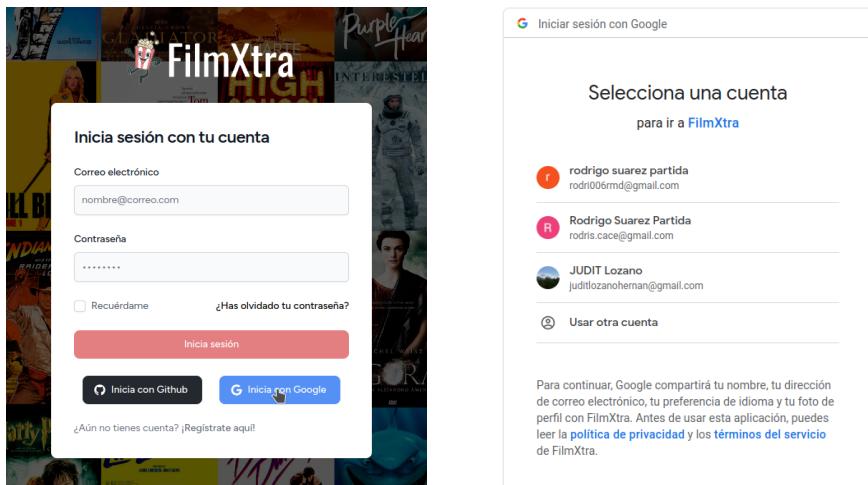
\*Nota: Se observó que al utilizar la etiqueta 'Link' proporcionada por Inertia para los enlaces, no funcionaba correctamente, pero al utilizar la etiqueta 'a' convencional sí.

```
Route::get('login-google', [SocialAuthController::class, 'redirectToProvider'])
    ->name('google.login');
```

Después de hacer clic en el botón, se realiza una solicitud GET a la ruta ubicada en 'Auth.php', la cual se dirige a la función 'redirectToProvider' en la clase 'SocialAuthController'.

```
    Predefined Overrides
public function redirectToProvider(): RedirectResponse
{
    return Socialite::driver('google')->redirect();
}
```

En esta parte, configuramos Socialite para utilizar el controlador 'google' que creamos anteriormente y le indicamos que utilice el redireccionamiento asociado, que en este caso es la siguiente ruta: <http://localhost:8000/auth/google/callback>. Esto permite que, después de seleccionar una cuenta de Google, se realice un redireccionamiento.



Ahora, una vez hemos realizado la configuración, seleccionamos la cuenta de Google que deseamos utilizar y se ejecuta el siguiente proceso.

```
Route::get('auth/google/callback', [SocialAuthController::class, 'handleCallback'])
    ->name('google.login.callback');
```

En este punto, se invoca la función 'handleCallback' de la clase 'SocialAuthController'.

```
public function handleCallback(): RedirectResponse
{
    try{
        $user = Socialite::driver('google')->user();
    } catch(\Exception $e) {
        return redirect('/login');
    }

    $userExist = User::where('google_id', $user->id)->first();

    if ($userExist !== null) {
        Auth::login($userExist);
    } else{
        $newUser = User::create([
            'email' => $user->email,
            'google_id' => $user->id,
            'password' => Hash::make($user->id),
            'email_verified_at' => Date::now()
        ]);
        $newUser->sendEmailPassword();

        Auth::login($newUser);
    }
    return redirect('/');
}
```

En primer lugar, se crea la variable \$user y se asigna el usuario devuelto por Google. A continuación, se realiza una consulta a la base de datos filtrando por el campo google\_id, que es único.

Se verifica si el usuario existe. En caso afirmativo, se autentica sin realizar más acciones. En caso contrario, se crea un nuevo usuario utilizando los datos de la variable \$user proporcionados por Google. Se destaca que en este punto se verifica automáticamente la validez del correo electrónico, ya que se asume que proviene de una fuente confiable.

Luego, se utiliza el campo google\_id como contraseña opcional (aunque se podría eliminar esta opción) y se autentica al nuevo usuario \$newUser. Y por último, se redirige al usuario a la ruta principal del proyecto.

## 7. Desarrollo de la base de datos

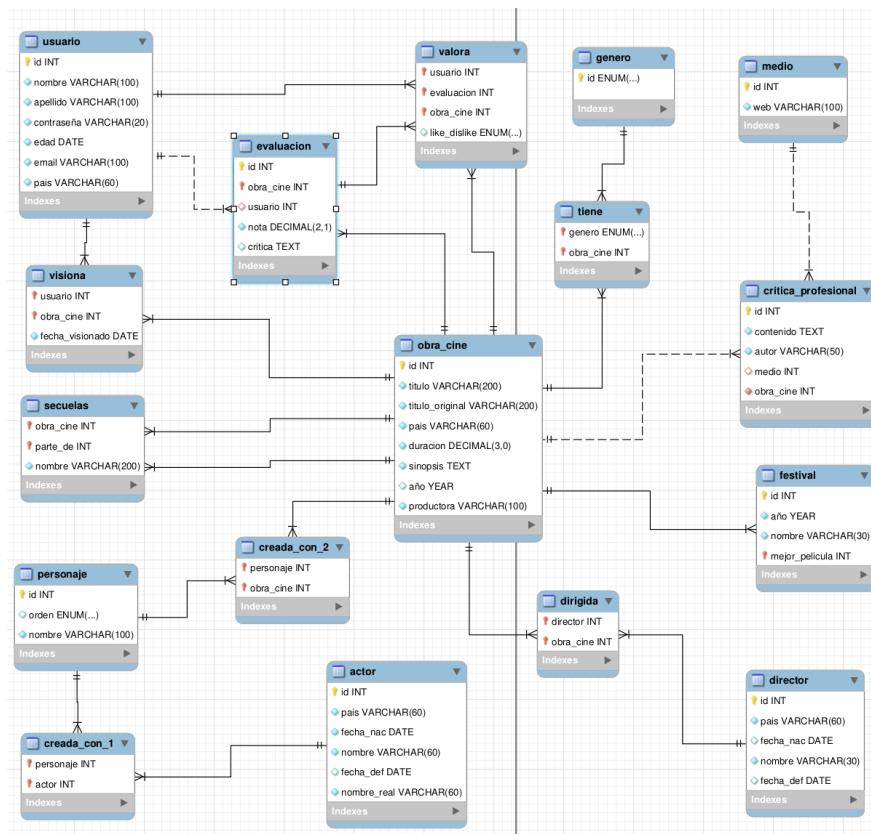
### 7.1. Adopción y diseño de la base de datos para la persistencia

Inicialmente, partimos de una base de datos desarrollada en un proyecto del año pasado en la asignatura “Bases de Datos”, durante el primer año de Desarrollo de Aplicaciones Web. Sin embargo, al comenzar el presente proyecto, se dedicó un tiempo considerable a modificarla. Estas modificaciones fueron realizadas para satisfacer las nuevas necesidades específicas de este proyecto o para mejorar aspectos deficientes. Durante estas modificaciones, se realizaron diversas acciones, como la inclusión de nuevas tablas, la modificación de columnas en algunas tablas, la adición de nuevas columnas y la modificación de relaciones entre las tablas.

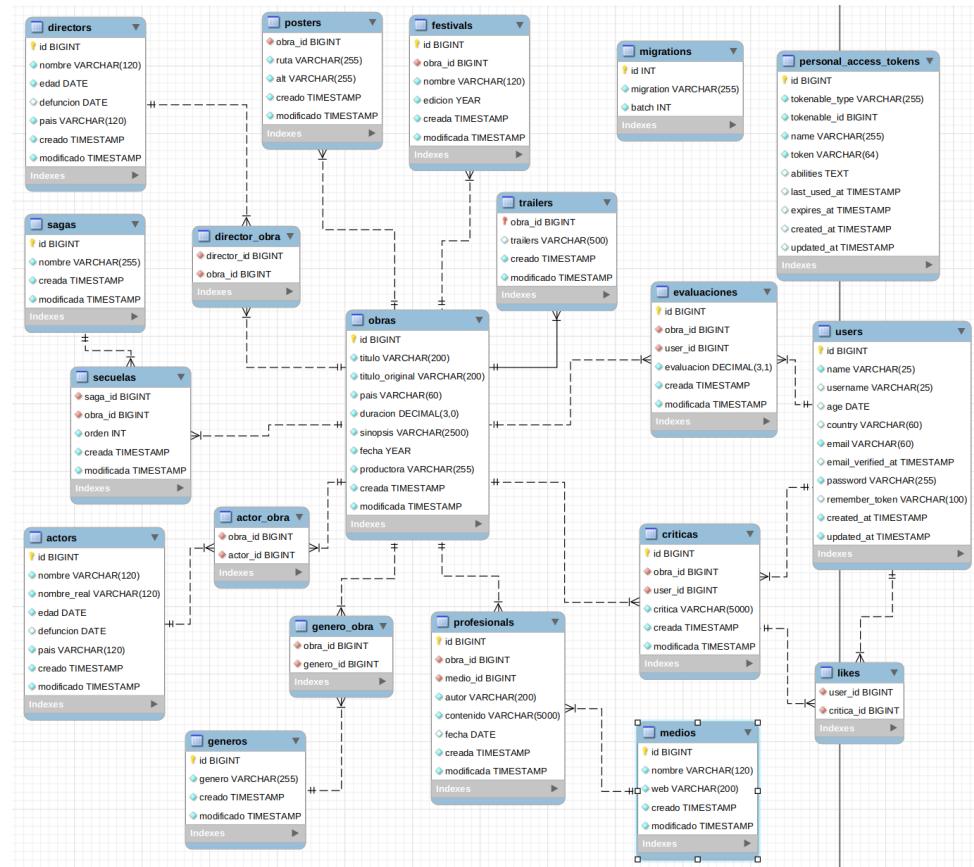
A medida que avanzábamos en el desarrollo del código del proyecto, surgieron nuevas necesidades. Por ejemplo, se tuvo que añadir una tabla para contener los enlaces de los trailers. Es importante mencionar que, en el futuro, es probable que sea necesario volver a realizar modificaciones en la base de datos para adaptarse a ideas y requerimientos adicionales del proyecto.

Un ejemplo de posibles modificaciones futuras sería la creación de una sección a la que solo las personas con el rol de gestores puedan acceder, con el fin de registrar nuevos datos en la capa de persistencia. En ese caso, sería necesario agregar una nueva columna, como 'rol', en la tabla de usuarios.

A continuación, veamos cómo lucía el diagrama de tablas original antes de adaptarlo al proyecto Filmxtra:



Actualmente, la base de datos ha sido modificada y adaptada para satisfacer las necesidades del proyecto Filmxtra. A continuación, presentamos el estado actual de la base de datos:



Durante el desarrollo del proyecto, se realizaron modificaciones importantes en la estructura de la base de datos para adaptarla adecuadamente a las necesidades de Filmxtra. A continuación, detallamos los cambios realizados:

- **Tablas Nuevas:** se crearon las tablas 'trailers' y 'posters' para almacenar información específica de los enlaces a los trailers y de los carteles de las películas.
- **Modificación de Relaciones:**
  - La relación '**obras-secuelas**' se reformuló y ahora se denomina '**obras-secuelas-sagas**'. Esta nueva estructura evita la redundancia de tuplas innecesarias y proporciona datos adicionales, como el orden de las secuelas dentro de una saga, incluso si una secuela forma parte de la saga pero no sigue el orden lineal.
  - La relación '**obras-actores-personajes**' se modificó eliminando la tabla 'personajes' y manteniendo únicamente la tabla 'actores', según las necesidades específicas del proyecto.
- **Eliminación de Tablas:** algunas tablas, como 'visiona', fueron eliminadas, ya que no eran relevantes para el proyecto actual.
- **Rediseño de la Relación entre 'obras', 'users', 'criticas' y 'evaluaciones':** para satisfacer las necesidades del proyecto, se rediseñó esta relación, incluyendo la tabla 'likes'. Esta tabla reemplazó una columna en la tabla 'evaluaciones' y proporciona independencia de datos: una crítica es diferente de una evaluación, y un

like no está relacionado con las anteriores. Por lo que un usuario puede crear cada uno de estos elementos de forma independiente.

- **Rediseño de la Relación 'n:m' entre 'obras' y 'géneros':** esta relación se ajustó para utilizar tuplas tradicionales con identificadores (IDs) en lugar de un 'enum'. Este cambio se realizó para solucionar problemas al generar el modelo y las migraciones en Laravel.

En general, consideramos que el resultado de estas modificaciones es coherente y tiene en cuenta posibles necesidades futuras del proyecto Filmxtra.

Un ejemplo concreto de las posibles modificaciones futuras que nos podríamos enfrentar es la implementación de un sistema de roles en el cual existan usuarios con el rol de "gestores", quienes podrían tener la capacidad de introducir datos en la base de datos a través de una interfaz gráfica. Para incorporar este sistema, podríamos añadir una nueva columna en la tabla 'users' o, para evitar tuplas nulas, crear una nueva tabla con una clave foránea que haga referencia a 'users' e incluya los usuarios con dicho rol.

Es importante destacar que, en general, resulta difícil anticipar todas las necesidades que podríamos tener en el futuro. Sin embargo, la decisión de separar los datos en varias tablas mediante diversas relaciones puede facilitar las modificaciones y adiciones posteriores en el sistema.

Además, se aplicó la normalización a la base de datos, incluso llegando en algunos casos hasta la cuarta forma normal (4FN). No obstante, debido a las restricciones de tiempo y las limitaciones de recursos, algunas de las modificaciones realizadas posteriormente pueden no haber alcanzado un nivel de normalización tan avanzado, quedándose en la primera forma normal (1NF) o la segunda forma normal (2FN).

## 7.2. Inserts. Información almacenada

Durante el desarrollo del proyecto actual, nos encontramos con la necesidad de modificar los bloques de "inserts" existentes en la base de datos, así como crear nuevos "inserts" para las nuevas tablas y relaciones. Por ejemplo, Elena se encargó de buscar y agregar los carteles de las películas.

Con el objetivo de agilizar el trabajo actual y futuro, también creamos una serie de procedimientos que utilizan cursor y variables temporales. Estos procedimientos, basados en datos de otras columnas como los títulos de las obras, insertan información en otras tablas, como la tabla de 'posters'. Durante este proceso, los títulos se adaptan y se eliminan caracteres innecesarios.

A continuación, puedes ver una captura de pantalla que muestra dos de estos procedimientos. El primero se utiliza para obtener el 'id' de la obra y utilizarlo como 'id' para los 'posters'. El segundo procedimiento, que se muestra en la misma captura de pantalla, se encarga de generar los campos 'ruta' y 'alt' en la tabla 'posters' basándose en los títulos de las obras.

```

/*
 * INSERTS PARA POSTERS */
/* Procedimiento para insertar ids existentes de obras en posters */
delimiter //
CREATE PROCEDURE ides()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE id_temp INT;
DECLARE c CURSOR FOR SELECT id FROM obras;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
OPEN c;
c_loop: LOOP
FETCH c INTO id_temp;
IF done THEN
LEAVE c_loop;
END IF;
insert into posters (obra_id, ruta, alt) VALUES (id_temp, '', '');
END LOOP;
CLOSE c;
END;
//
delimiter ;
call ides();
drop procedure ides;

/*
 * Procedimiento para modificación de campos ruta y alt */
delimiter //
CREATE PROCEDURE miReemplazo()
BEGIN
DECLARE done INT DEFAULT FALSE;
/* Variables que vamos a manejar en el procedimiento */
DECLARE id_temp INT;
DECLARE modificar_temp CHAR(255); /* No utilizada pero ej. por si se quiere usar en las modificaciones */
/* Cursor con la consulta para obtener campos necesarios para modificar de la tabla */
DECLARE c CURSOR FOR SELECT obra_id, ruta FROM posters;
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
OPEN c;
c_loop: LOOP
/* Asignamos a las variables los valores obtenidos del cursor */
FETCH c INTO id_temp, modificar_temp;
IF done THEN
LEAVE c_loop;
END IF;
/* Aplicamos modificaciones */
update posters set ruta = (select titulo_original from obras where id = id_temp) where id_temp = obra_id;
update posters set ruta = replace(ruta, '.', '_') where id_temp = obra_id;
update posters set ruta = replace(ruta, ':', '_') where id_temp = obra_id;
update posters set ruta = replace(ruta, ',', '') where id_temp = obra_id;
update posters set ruta = replace(ruta, '..', '_') where id_temp = obra_id;
update posters set ruta = replace(ruta, '\\', '_') where id_temp = obra_id;
update posters set ruta = concat('Poster de ', '\\', (select titulo_original from obras where obra_id = obras.id), '\\') where id_temp = obra_id;
END LOOP;
CLOSE c;
END;
//
delimiter ;
call miReemplazo();
drop procedure miReemplazo;

```

## 8. Conclusión

La combinación de tecnologías, entornos y lenguajes de programación que hemos utilizado en el proyecto FilmXtra nos ha permitido lograr un resultado que supera nuestras expectativas iniciales. Estas herramientas nos han brindado la capacidad de desarrollar una web de cine excepcional, que destaca por su creatividad y funcionalidad.

A lo largo de este proyecto, hemos experimentado un proceso de creación en el que hemos investigado, probado y refinado constantemente nuestras ideas. Cada integrante del grupo ha contribuido con su conocimiento y habilidades, lo que ha llevado a la consecución de un producto final excepcional.

Nos sentimos orgullosos de haber creado una web de cine única, considerando que somos estudiantes de segundo año del Grado Superior de Desarrollo de Aplicaciones Web. Esta experiencia nos ha permitido aplicar nuestros conocimientos teóricos en un proyecto práctico, poniendo en práctica nuestras habilidades de programación y diseño.

En resumen, este proyecto ha sido un verdadero desafío, pero también una gran oportunidad de aprendizaje. Hemos adquirido experiencia en la creación y desarrollo de una web de cine, desde la planificación inicial hasta la implementación final. Estamos

emocionados de documentar todo el proceso, incluyendo nuestra investigación, pruebas y logros, con el objetivo de presentar la mejor web de cine jamás creada por estudiantes de segundo año de Desarrollo de Aplicaciones Web.

## 9. Proyectos a futuro

A continuación, se presentan algunas de las tareas y mejoras que teníamos en mente para FilmXtra, pero debido a limitaciones de tiempo, aún no hemos podido implementarlas. Estas son ideas que dejamos para el futuro, con el objetivo de seguir mejorando y expandiendo la funcionalidad de nuestro sitio web.

### 9.1. Organización del código

Nuestra prioridad inicial es refactorizar y optimizar el código, garantizando su organización y eliminando duplicaciones y código desordenado. Es fundamental establecer una base sólida desde el principio para asegurar la sostenibilidad a largo plazo del proyecto.

### 9.2. Podcast

Deseamos integrar un sistema de podcast en el diseño persistente del sitio web, para que la reproducción del podcast se mantenga mientras navegas por diferentes páginas.

### 9.3. Foto de usuario

Queremos permitir a los usuarios de FilmXtra que puedan agregar o editar una foto de perfil, siempre y cuando estén registrados.

### 9.4. Listas

Tenemos planeado incorporar listas de películas creadas por los administradores, así como dar la oportunidad a los usuarios de crear y compartir sus propias listas. Esto les permitirá consultar las listas de otros usuarios y obtener ideas.

### 9.5. Honeypot

Incluir un honeypot, un sistema que agrega un campo oculto al formulario de registro de usuarios. Si ese campo se completa, significa que un bot intenta realizar registros automatizados. Esta herramienta de seguridad informática nos ayudará a prevenir posibles ataques.

### 9.6. Mejora del botón de búsqueda

Trabajaremos en mejorar la funcionalidad y usabilidad del botón de búsqueda.

## 9.7. Añadir fotos de las películas

Para aumentar el interés y la identificación de los usuarios con las películas, deseamos incluir fotos o capturas de pantalla tomadas durante la filmación, similar a cuando se muestran imágenes de un videojuego antes de su compra.

## 9.8. Valoraciones y críticas de usuarios

Implementaremos un sistema que permita a los usuarios editar y visualizar todas sus evaluaciones y críticas en su página de cuenta.

## 9.9. Segundo tráiler

Habilitaremos la opción de agregar un segundo tráiler a las películas. Actualmente, solo se puede ver un tráiler en la base de datos, pero al separarlos con un punto y coma en el insert, podremos crear un array con los nombres de cada tráiler, gracias al método split de MySQL.

## 9.10. Roles

Crearemos diferentes roles para la página web, con el objetivo de permitir a los gestores del sitio realizar tareas de forma más intuitiva a través de una interfaz gráfica amigable.

## 9.11. Notificaciones

Incluiríremos notificaciones vía SMS o email para informar a los usuarios cuando reciben likes en sus críticas u otras interacciones con otros usuarios de la web. Y además, añadiremos nuevos mensajes interactivos en toda la web, como un mensaje emergente cuando un usuario se registra o inicia sesión en FilmXtra.

Además, el PopUp de cierre se sesión, inicio de sesión y registro ya están diseñados, sólo habría que implementarlos.

```

<template>
  <div id="info-popup" tabindex="-1" class="hidden overflow-y-auto overflow-x-hidden fixed top-0 right-0 left-0 z-50 w-full h-full">
    <div class="relative p-4 w-full max-w-tg h-full md:h-auto">
      <div class="mb-4 text-sm font-light text-gray-500">
        <h3 class="mb-3 text-2xl font-bold text-gray-900">Has cerrado sesión en FilmXtra correctamente!</h3>
        <p>Tu sesión de FilmXtra se ha cerrado correctamente, esperamos que vuelvas pronto. ¡En FilmXtra te queremos!</p>
      </div>
      <div class="justify-between items-center pt-0 sm:flex sm:space-y-0">
        <div class="items-center space-y-4 sm:space-x-4 sm:flex sm:space-y-0">
          <button id="close-modal" type="button" class="py-2 px-4 w-full text-sm font-medium text-gray-500 rounded-lg border border-gray-300 hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-gray-300 transition-all">Cerrar</button>
          <button id="confirm-button" type="button" class="py-2 px-4 w-full text-sm font-medium text-gray-500 rounded-lg border border-gray-300 hover:bg-gray-50 focus:outline-none focus:ring-2 focus:ring-gray-300 transition-all">Continuar</button>
        </div>
      </div>
    </div>
  </div>
</template>

```

## 9.12. Laravel Policies

Aprovecharemos la funcionalidad proporcionada por Laravel para controlar el acceso a los recursos de nuestra aplicación. Utilizaremos esta característica como medida de seguridad adicional, por ejemplo, al permitir que solo el usuario que escribió una crítica pueda modificarla.

## 9.13. Información sobre cines

Agregaremos información sobre cines, próximas películas y la cartelera actual.

## 9.14. Despliegue

Realizaremos pruebas de despliegue del sitio web para asegurarnos de su correcto funcionamiento en entornos de producción.

## 9.15. Ordenamiento

Permitiremos a los usuarios ordenar las críticas y películas por número de likes, fecha, y otras opciones como orden alfabético, cantidad de críticas, puntuación más baja, etcétera.

## 9.16. Sesiones

Implementaremos un sistema de sesiones en la base de datos para rastrear el número de visitantes en el sitio web. También consideraremos la incorporación de paquetes de código abierto, como Laravel Visits, para almacenar información detallada de las visitas, como la dirección IP, el país y el navegador utilizado. Estos datos nos ayudarán a auditar el perfil de los usuarios que nos visitan.

```
composer require awssat/laravel-visits
```

## 9.17. Tienda

Como se mencionó en las ideas iniciales del proyecto, planeamos implementar una tienda online que ofrezca productos relacionados con el cine, como una forma adicional de monetización junto con la publicidad.

## 9.18. Inicio de sesión con GitHub

Añadir autenticación con GitHub al igual que con Google.

## 9.19. Introducir más películas

Incrementar la variedad de películas y considerar la posibilidad de incluir series en un futuro.

## 9.20. Páginas de error

Configurar las páginas de error para mostrar mensajes personalizados, no limitándose solo a la página de error 404. Laravel facilita esto mediante un bloque en la clase "Handler" que te permite personalizar las clases de errores y sus mensajes.

```
protected array $messages = [
    500 => 'Oh oh, algo ha debido ir muy mal',
    503 => 'Servicio no disponible',
    404 => 'No te montes películas, la página que buscas no existe...',
    403 => 'No estás autorizado para esto',
];
```

En resumen, FilmXtra tiene grandes planes para el futuro. Aunque existen algunas tareas pendientes debido a la falta de tiempo, estaremos comprometidos a seguir desarrollando y mejorando nuestro sitio web si tenemos la oportunidad.

## 10. Tareas FilmXtra

Como equipo de trabajo, comprendemos la importancia de documentar nuestras actividades diariamente, por ello, en este proyecto, adoptamos la idea de utilizar una ficha semanal similar a las que utilizamos en nuestras prácticas en los centros de trabajo. Para ello, creamos una hoja de cálculo en Google compartida, en la cual íbamos registrando nuestras tareas y avances diarios. A continuación, mostramos los resultados que hemos obtenido a lo largo de estos meses:

			<b>PLANIFICACIÓN FILMXTRA</b>		Grupo	Víctor	Rodri	Elena	Tutoría
<b>MARZO</b>									
LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO	DOMINGO			
		1	2	3	4	5			
6	7	8	9	10	11	12			
13	14	15	16	17	18	19			
20	21	22	23	24	25	26			
27	28	29	30	31					
	Temática y 1º versión anteproyecto	Envío anteproyecto							
		Revisión y envío del anteproyecto							

ABRIL						
LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO	DOMINGO
					1	2
						Realización prototipos webs a papel
3	4	5	6	7	8	9
				Elección nombre proyecto		
Realización prototipos webs a papel	Búsqueda nombre proyecto				Realización prototipos webs Figma	Realización logotipos en Canva
10	11	12	13	14	15	16
Votación logo, colores y tipografía	Instalar PHPStorm & licencia	CONF phpstorm	Instalación PhpStorm con Victor		Planteamiento y división trabajo Front	Creación repositorio
17	18	19	20	21	22	23
Reunión Laravel	Andamiaje proyecto laravel		Creación rama development		Crear modelos migraciones controlados	Reestructuración andamiaje & breeze
Tutoría						
24	25	26	27	28	29	30
Reorganizacion vistas breeze			Finalizadas migraciones y modelos	Finalizados Inserts	Probadas migraciones & inserts	
		Investigación de proyecto prueba Laravel + Vón sobre la posibilidad de introducir v	Investigación Wordpress en Laravel	Descarga posters y subida a GitHub	Modificación logo	Realización página inicio Bootstrap

MAYO							
LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO	DOMINGO	
1	2	3	4	5	6	7	
Boceto inicial de rutas	Creación de rutas iniciales	Reunión Laravel	Reunión Vue y Tailwind	Reunión Vue y Tailwind	Reunión PHPStorm y Tailwind	Reunión PHPStorm y Tailwind	
Info tailwind y creación componentes	Organización de componentes	Implementación Flowbite	Test y correr rutas/vistas iniciales	Implementación layout persistente			
Formación Tailwind y página inicio	Envío página inicio de Tailwind	Info e implantación comp dinámicos	Resolución de problemas	Investig variables email y crear correo	Testeo envío emails y configuración	Pruebas en proyecto de ejemplo	
8	9	10	11	12	13	14	
Conf array rutas error	Implementacion dayjs	Finalizada vista bienvenida	Componentes: poster, form, pag	Componentes: poster, form, pag	Finalizado formulario filtrado	Optim y refact código (estilo y resto)	
Implementación email proyecto real	Resolución errores implementación	Nuevo error en autenticación	Solución 'rápida' a problema autenticación	Cambio plan verif 2 pasos a sesiones		Fin implementación sesiones y nuevo error en verificación	
Realización página filtrar por género	Realización página error 404	Página error 404 y modificaciones	Formulario filtrar y página error 404	Página error 404 y logo error 404			
15	16	17	18	19	20	21	
Finalizada paginación	Finalizado formulario filtrado	Pasando todos datos ficha (prov)	Mejor comp.poster+dinam prop título	Todos los formularios rematados	Creado SFC form filtrado emita form padre y propia de consulta	Like, nota media y Obra.vue funciona	
Estudiando las rutas para ver de donde procedía el error	Errores resueltos (a priori)	Merge con rama principal y desconfiguración	Solucionar errores de merge	Fallo con sesiones Login	Arreglado logueo con sesiones	Nuevo error 'Duplicate entry error'	
Página error 404	Página login, register y reset pass	Página obra	Página obra y carpeta auth	Página cuenta, valoraciones y obra	Página valoraciones y obra	Inserts críticas profesionales	
22	23	24	25	26	27	28	
Enum actor/direct y aviso logeo like	Estrellitas, nota media, redirect logeo	Reorg obra, trailers y orden top val	Contr y vista fichaValo y form crit/valo	Valid form crit/val y crear control pag	InfoController para funciones globales	Pop ups críticas/evaluaciones	
Búsqueda información y testeos	Prueba de implementación github + configurarlo	Login para que no salte error al venir	Pruebas para solucionar duplicate entry error	Duplicate entry error, búsqueda información en foros	Prueba eliminar atributo sesión para arreglar verif correo		
Inserts críticas profesionales	Inserts likes y críticas usuarios	Inserts trailers y primeros retoques	Inserts críticas y retoques	Estilos formularios valoraciones	Estilos formularios valoraciones		
29	30	31					
Tutoría							
Form crit/evas y mensj dif crítica mod	Gifs aleatorios para los pop ups	Estilos y creación básica buscador					
Información Twilio y su implementación	Volviendo a modelo default de verif 2 pasos y adaptarlo	Implementación Google OAuth y vídeos					
	Descarga GIFs e inicio memoria	Realización de la memoria y revisar					

JUNIO						
LUNES	MARTES	MIÉRCOLES	JUEVES	VIERNES	SÁBADO	DOMINGO
			1 Reunión migración verif correo Correcciones y form crit/evas Realización de la memoria Realización de la memoria y revisar	2 Mas correcciones Realización de la memoria Realización de la memoria	3 Realización de la memoria Realización de la memoria	4 Realización de la memoria Realización de la memoria
5 Tutoría Realización de la memoria	6 Preparación presentación Realización de la memoria	7 Entrega TFG 23:59h Preparación presentación Preparación presentación	8 Preparación presentación Preparación presentación Preparación presentación	9 Preparación presentación Preparación presentación	10 Preparación presentación	11 Preparación presentación
12 Defensa TFG 14:00h	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30		

# 11. Bibliografía

## 11.1. Documentación para la memoria

1. **Laravel:** colaboradores de Wikipedia. (2023). Laravel. Wikipedia, la encyclopédie libre. <https://es.wikipedia.org/wiki/Laravel>
2. **Tailwind CSS:** colaboradores de Wikipedia. (2023c). Tailwind CSS. Wikipedia, la encyclopédie libre. [https://es.wikipedia.org/wiki/Tailwind\\_CSS](https://es.wikipedia.org/wiki/Tailwind_CSS)
3. **Flowbite:** M.Domínguez. (2022, 9 febrero). Componentes de Tailwind CSS: Cómo empezar con Flowbite 2021. TodoXampp. <https://todoxampp.com/componentes-de-tailwind-css-flowbite/>
4. **MySQL:** colaboradores de Wikipedia. (2023). MySQL. Wikipedia, la encyclopédie libre. <https://es.wikipedia.org/wiki/MySQL>
5. **MySQL Workbench:** colaboradores de Wikipedia. (2023). MySQL Workbench. Wikipedia, la encyclopédie libre. [https://es.wikipedia.org/wiki/MySQL\\_Workbench](https://es.wikipedia.org/wiki/MySQL_Workbench)
6. **JetBrains Toolbox:** ¿Qué es Toolbox de JetBrains? (s. f.). Preguntas frecuentes sobre licencias y compras. <https://sales.jetbrains.com/hc/es/articles/206544639--Qu%C3%A9-es-Toolbox-de-JetBrains->
7. **PhpStorm:** PhpStorm: IDE de PHP y editor de código de JetBrains. (2021, 2 junio). JetBrains. <https://www.jetbrains.com/es-es/phpstorm/>
8. **Visual Studio Code:** colaboradores de Wikipedia. (2023e). Visual Studio Code. [https://es.wikipedia.org/wiki/Visual\\_Studio\\_Code](https://es.wikipedia.org/wiki/Visual_Studio_Code)
9. **GitHub:** colaboradores de Wikipedia. (2023c). GitHub. Wikipedia, la encyclopédie libre. <https://es.wikipedia.org/wiki/GitHub>
10. **Chrome:** By. (2016, 26 julio). 10 herramientas de Chrome que todo desarrollador va a amar - Tech blog for developers | Facilcloud. Tech blog for developers Facilcloud. <https://www.facilcloud.com/noticias/10-herramientas-de-chrome-que-todo-desarrollador-va-a-amar/>
11. **HTML5:** Admin. (2022, 16 mayo). HTML5, ¿qué es y para qué sirve? Mi Formación Gratis. <https://www.miformaciongratis.com/blog-post/html5-que-es-y-para-que-sirve/>
12. **CSS3:** Santos, D. (2023, 16 mayo). Introducción al CSS: qué es, para qué sirve y otras 10 preguntas frecuentes. Hubspot. <https://blog.hubspot.es/website/que-es-css>
13. **ChatGPT:** colaboradores de Wikipedia. (2023). ChatGPT. Wikipedia, la encyclopédie libre. <https://es.wikipedia.org/wiki/ChatGPT>
14. **Bootstrap:** 7 razones para usar Bootstrap en tu web. (s. f.). Immune Technology Institute. <https://immune.institute/blog/razones-usar-bootstrap-web/>
15. **JavaScript:** colaboradores de Wikipedia. (2023). JavaScript. Wikipedia, la encyclopédie libre. <https://es.wikipedia.org/wiki/JavaScript>
16. **Vue.js:** colaboradores de Wikipedia. (2023). Vue.js. Wikipedia, la encyclopédie libre. <https://es.wikipedia.org/wiki/Vue.js>
17. **Scribbr:** Scribbr. (2022, 28 noviembre). Formato APA con el Generador APA de Scribbr. <https://www.scribbr.es/citar/generador/apa/>

## 11.2. Documentación del back-end (Víctor)

1. **Documentación de Laravel:** *Laravel - The PHP Framework For Web Artisans.* (s. f.-c). <https://laravel.com/>
2. **Stack Overflow:** *Stack Overflow - Where Developers Learn, Share, & Build Careers.* (s. f.-b). Stack Overflow. <https://stackoverflow.com/>

## 11.3. Documentación del diseño web (Elena)

1. **Documentación Tailwind:** *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML.* Tailwind CSS. <https://tailwindcss.com/>
2. **Footer Bootstrap:** *Bootstrap 5 Responsive Footer Section Example.* [https://frontendshape.com/post/bootstrap-5-responsive-footer-section-example?utm\\_content=cmp-true](https://frontendshape.com/post/bootstrap-5-responsive-footer-section-example?utm_content=cmp-true)
3. **Navbar Bootstrap:** Contributors, M. O. J. T. A. B. (s. f.). *Navbar.* <https://getbootstrap.com/docs/5.3/components/navbar/#responsive-behaviors>
4. **Navbar Tailwind:** Themesberg. (s. f.). *Tailwind CSS Navbar - Flowbite.* <https://flowbite.com/docs/components/navbar/>
5. **Footer Tailwind:** Themesberg. (s. f.-a). *Tailwind CSS Footer - Flowbite.* <https://flowbite.com/docs/components/footer/>
6. **Carrusel Tailwind:** Themesberg. (s. f.-a). *Tailwind CSS Carousel - Flowbite.* <https://flowbite.com/docs/components/carousel/>
7. **Página 404 Tailwind:** *404 Pages - Official Tailwind CSS UI Components.* (s. f.). <https://tailwindui.com/components/marketing/feedback/404-pages>
8. **Text Area Tailwind:** Themesberg. (s. f.-d). *Tailwind CSS Textarea - Flowbite.* <https://flowbite.com/docs/forms/textarea/>
9. **Popup Tailwind:** *Tailwind CSS Popup - Flowbite.* (s. f.). Tailwind CSS. Recuperado 31 de mayo de 2023, de <https://flowbite.com/blocks/marketing/popup/>
10. **Formulario Reset Password Tailwind:** *Tailwind CSS Reset Password Form - Flowbite.* (s. f.). Tailwind CSS. Recuperado 31 de mayo de 2023, de <https://flowbite.com/blocks/marketing/reset-password/>
11. **Formulario Login Tailwind:** *Tailwind CSS Login Form - Flowbite.* (s. f.). Tailwind CSS. Recuperado 31 de mayo de 2023, de <https://flowbite.com/blocks/marketing/login/>
12. **Formulario Register Tailwind:** *Tailwind CSS Register Form - Flowbite.* (s. f.). Tailwind CSS. Recuperado 31 de mayo de 2023, de <https://flowbite.com/blocks/marketing/register/>
13. **Tailwind grid responsive:** Bongers, C. (2021, 24 abril). *Tailwind grid responsive 4 column blocks - Daily Dev Tips.* Daily Dev Tips. <https://daily-dev-tips.com/posts/tailwind-grid-responsive-4-column-blocks/>
14. **Diseño responsive Tailwind:** Estecnologico. (2022, 9 septiembre). *Diseño RESPONSIVE - Fácil - Curso TAILWIND 2022 GRATIS [Vídeo].* YouTube. <https://www.youtube.com/watch?v=ImDrBYXStSI>
15. **Grid Responsive Tailwind:** CorneGramm Desarrollo web - Web development. (2021, 27 marzo). *GRID Tailwind CSS Responsive Tutorial 2021 [Vídeo].* YouTube. <https://www.youtube.com/watch?v=nMMUkHgJCwo>

16. **Carrusel Bootstrap:** Kiko Palomares. (2022, 27 julio). *Cómo hacer un CARRUSEL Automático de IMÁGENES con BOOTSTRAP 5* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=mx5ay2pDopk>
17. **Slider Bootstrap:** Tecno Código. (2021, 11 mayo). *Crear slider o carrusel de imágenes con [Bootstrap]* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=AFUx77QPPBI>
18. **Aprender Bootstrap:** freeCodeCamp Español. (2022, 10 noviembre). *Aprende Bootstrap 5 - Curso de Bootstrap desde cero* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=QCw0L6FupQ0>
19. **Filmaffinity:** FilmAffinity. (s. f.). FilmAffinity. <https://www.filmaffinity.com/es/main.html>
20. **Flechas UTF-8:** [https://www.w3bai.com/es/charsets/ref\\_utf\\_arrows.html](https://www.w3bai.com/es/charsets/ref_utf_arrows.html)
21. **ChatGPT:** ChatGPT. (s. f.). ChatGPT. <https://chat.openai.com/>
22. **Canva:** Canva. Free Design Tool: <https://www.canva.com/>
23. **Página error 404:** 404 Pages - Official Tailwind CSS UI Components. (s. f.-b). <https://tailwindui.com/components/marketing/feedback/404-pages>
24. **Google Sheets:** Google Sheets: Sign-in. (s. f.). <https://docs.google.com/spreadsheets/u/0/>
25. **Google Docs:** Google Docs: Sign-in. (s. f.). <https://docs.google.com/document/u/0/>

## 11.4. Documentación de redes (Rodrigo)

1. **Documentación Socialité:** Laravel Socialite para logueo con Google y otras plataformas *Laravel - The PHP Framework For Web Artisans*. (s. f.). <https://laravel.com/docs/10.x/socialite>
2. **Logueo con Google:** durante el proceso de implementación del inicio de sesión con Google y la configuración de la consola de Google Cloud, encontré varios vídeos que resultaron especialmente útiles. Estos vídeos me brindaron las instrucciones necesarias y los pasos a seguir para llevar a cabo estas tareas de manera efectiva. A continuación, mencionaré algunos de los vídeos que me resultaron más útiles:
  - Marc Garcia - Magarrent. (2021, 14 agosto). *Iniciar sesión y registrar usuarios con Google Auth & Laravel Socialité [Guía Paso a Paso]* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=BXXmboZRLYo>
  - Takneeki Code. (2022, 30 julio). *LARAVEL SOCIALITE - LOGIN WITH SOCIAL ACCOUNT - GOOGLE SOCIAL LOGIN* [Vídeo]. YouTube. [https://www.youtube.com/watch?v=j-lVevL\\_72E](https://www.youtube.com/watch?v=j-lVevL_72E)
  - Code With Tony. (2023, 9 febrero). *Laravel Socialite Login with Google and Github | Custom Laravel Breeze* [Vídeo]. YouTube. <https://www.youtube.com/watch?v=XFmJfTzGqzY>
3. **Resolución problemas:** cuando me enfrentaba a desafíos o tenía dudas durante el desarrollo del proyecto, solía recurrir a Stack Overflow como una valiosa fuente de ayuda. La comunidad de Stack Overflow ha sido de gran utilidad en múltiples ocasiones, ya que ofrece respuestas y soluciones a problemas comunes y específicos que pueden surgir durante el proceso de desarrollo. *Stack Overflow - Where Developers Learn, Share, & Build Careers*. (s. f.). Stack Overflow. <https://stackoverflow.com/>

4. **Errores frecuentes:** hay errores que te pueden llevar más o meno, pero sin duda uno de los que me llevó a tener que invertir bastante tiempo fue este: Larainfo. *How to Solve SQLSTATE[23000] Integrity Constraint 1062 Error Laravel.*  
[https://larainfo.com/blogs/how-to-solve-sqlstate23000-integrity-constraint-1062-error-laravel?utm\\_content=cmp-true](https://larainfo.com/blogs/how-to-solve-sqlstate23000-integrity-constraint-1062-error-laravel?utm_content=cmp-true)
5. **Documentación sesiones:** para el manejo de sesiones con la verificación en dos pasos: *Laravel - The PHP Framework For Web Artisans.* (s. f.-b).  
<https://laravel.com/docs/10.x/session#main-content>
6. **Configuración Google cloud:** para la configuración de las variables de entorno de Google OAuth tuve que acudir a esta página *Google Cloud Platform.* (s. f.).  
<https://console.cloud.google.com/>
7. **Activar verificación en dos pasos:** para configurar el correo y permitir que el servidor de Google lo use para mandar emails *Sign in - Google Accounts.*  
[https://myaccount.google.com/signinoptions/two-step-verification?rapt=AElHL4M1fcL2pE\\_tbzNCJajv0\\_NV-lcv6fhQr2GJKjogxQK19\\_ptxdH79Hz5X7o0GsONyTiHrvL6K4wQC\\_AFiX8ZdJ5XLW4xMQ](https://myaccount.google.com/signinoptions/two-step-verification?rapt=AElHL4M1fcL2pE_tbzNCJajv0_NV-lcv6fhQr2GJKjogxQK19_ptxdH79Hz5X7o0GsONyTiHrvL6K4wQC_AFiX8ZdJ5XLW4xMQ)