

Universidad Tecnológica del Perú  
Ingeniería de Software  
Escuela de Ingeniería



"Sistema web de gestión de reservas y ventas de  
películas"

Avance de Proyecto que como parte del curso Desarrollo  
Web Integrado

Presentan los alumnos:

Aulla Gonzales, Jacson Michael  
Bellido de la Cruz, Samuel  
Gomez Zevallos, Sebastian Luis  
Quispe Arela, Sofia Denisse

Docente:

Jose Antonio Espinal Teves

Lima - Perú

2025

## **DEDICATORIA**

Le dedicamos este trabajo a nuestros familiares, amigos y compañeros, quienes nos han ayudado a alcanzar nuestros objetivos, apoyándonos y aconsejándonos en circunstancias poco favorables.

## AGRADECIMIENTOS

Expresamos nuestro sincero agradecimiento a nuestra docente guía, **Jose Antonio Espinal Teves**, por su constante apoyo, orientación y dedicación durante el desarrollo de este proyecto.

# Índice

<b>1 Introducción</b>	<b>6</b>
1.1 Planteamiento del Problema	6
1.2 Solución Propuesta	6
1.2.1 Descripción general	6
1.2.2 Diagrama de la aplicación	7
1.3 Objetivos	7
1.3.1 Objetivos generales	7
1.3.2 Objetivos técnicos	8
1.3.3 Objetivos personales	8
1.4 Planificación del trabajo	9
<b>2 Estado del Arte</b>	<b>10</b>
2.1 Tecnologías	10
2.1.1 Tecnologías para el desarrollo de APIs REST	10
2.1.1.1 Concepto de un API REST	10
2.1.2 Tecnologías para el desarrollo del FrontEnd y BackEnd	11
2.1.2.1 Frontend (HTML, CSS, Bootstrap, Thymeleaf)	11
2.1.2.2 Backend (Java Maven, JSP, Sprint Boot)	12
2.1.2.3 Base de Datos (MySQL)	12
2.1.3 Tecnologías para la extracción automática de información	13
2.1.4 Otras tecnologías	14
2.2 Fuentes de datos	14
2.2.1 Fuentes de información	14
2.2.2 Webs dedicadas al streaming de películas	16
2.2.3 Análisis de lo estudiado	16
<b>3 Desarrollo</b>	<b>17</b>

3.1	Diseño de la base de datos . . . . .	17
3.2	BackEnd . . . . .	18
3.2.1	Entidades sencillas . . . . .	19
3.2.2	Entidades Complejas . . . . .	22
3.3	Seguridad de la API . . . . .	26
3.4	FrontEnd . . . . .	28

# Índice de figuras

1	Diagrama de la arquitectura de la aplicación . . . . .	7
2	Vista de la api response . . . . .	11
3	Framework Bootstrap y motor de plantillas Thymeleaf . . . . .	11
4	Initializr de Spring Boot . . . . .	12
5	Vista de la base de datos MySQL . . . . .	13
6	clase pelicula donde estaran los datos . . . . .	13
7	Vista de la aplicacion de la fuente 1 . . . . .	15
8	Vista de la aplicacion de la fuente 2 . . . . .	15
9	Diagrama de la Base de Datos . . . . .	17
10	EndPoints del BackEnd . . . . .	18
11	Diagrama de la Base de Datos . . . . .	18
12	Diagrama de la Base de Datos . . . . .	19
13	FrontEnd del proyecto . . . . .	28
14	FrontEnd del proyecto . . . . .	29
15	FrontEnd del proyecto . . . . .	29
16	FrontEnd del proyecto . . . . .	30
17	FrontEnd del proyecto . . . . .	30
18	FrontEnd del proyecto . . . . .	31
19	FrontEnd del proyecto . . . . .	31
20	FrontEnd del proyecto . . . . .	32

## 1 Introducción

link del video: <https://youtu.be/tHVTNt62hGA>

### 1.1 Planteamiento del Problema

El acceso al entretenimiento digital, en especial a las películas, ha cambiado drásticamente en los últimos años. Aunque existen múltiples plataformas de streaming, muchos usuarios enfrentan limitaciones: catálogos restringidos, altos costos de suscripción o la imposibilidad de acceder únicamente a las películas que desean ver sin pagar por un paquete completo. Además, pequeños proveedores o negocios locales carecen de herramientas tecnológicas para ofrecer un sistema de reservas o alquileres en línea que sea eficiente y competitivo.

La ausencia de una plataforma accesible y sencilla para reservar películas en línea genera frustración en los usuarios, quienes deben recurrir a soluciones informales, poco seguras o dependientes de terceros. A su vez, los administradores no cuentan con un sistema automatizado para gestionar el catálogo, controlar el stock o llevar el registro de ventas y alquileres. Esto provoca pérdidas de clientes, baja escalabilidad y poca transparencia en la gestión.

### 1.2 Solución Propuesta

#### 1.2.1. Descripción general

Como respuesta a la problemática identificada, se plantea el desarrollo de una aplicación web de reservas de películas que funcione como un punto centralizado para usuarios y administradores.

El sistema permitirá a los usuarios registrarse, consultar el catálogo de películas dispo-

nibles, realizar reservas o compras y llevar un historial de sus operaciones, todo ello a través de una interfaz sencilla y accesible desde cualquier dispositivo con conexión a internet.

A su vez, la aplicación contará con un área administrativa que facilitará la gestión del catálogo, el control de ventas y el seguimiento de usuarios registrados, asegurando que la administración sea eficiente y organizada.

De esta manera, se busca brindar una solución integral que automatice el proceso de reserva y compra de películas, evitando las limitaciones de los sistemas tradicionales y ofreciendo una experiencia digital confiable, segura y moderna.

### 1.2.2. Diagrama de la aplicación

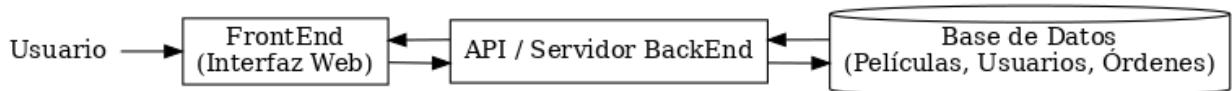


Figura 1: Diagrama de la arquitectura de la aplicación

## 1.3 Objetivos

### 1.3.1. Objetivos generales

- Desarrollar una plataforma web **StreamsUTP** que permita a los usuarios explorar, alquilar y comprar películas de forma segura y sencilla.
- Proporcionar un entorno administrativo que facilite la gestión de usuarios, películas y transacciones dentro del sistema.
- Ofrecer una experiencia de usuario clara y responsive, accesible desde cualquier dispositivo.

### 1.3.2. Objetivos técnicos

- Diseñar e implementar una base de datos en **MySQL** para manejar información de usuarios, películas, planes de suscripción y registros de visualización.
- Construir un **Backend con Spring Boot** que exponga servicios REST para el manejo de usuarios, catálogo y pagos.
- Integrar un **Frontend en Thymeleaf, HTML, CSS y JavaScript** que consuma la API, mostrando un catálogo dinámico y una interfaz responsiva.
- Configurar autenticación y autorización con **Spring Security + JWT**, asegurando que solo los usuarios registrados accedan al contenido según su plan.
- Implementar procesos de prueba (unitarias y de integración) que garanticen estabilidad y escalabilidad.

### 1.3.3. Objetivos personales

- Mejorar habilidades de trabajo colaborativo, control de versiones con GitHub y despliegue en entornos locales (XAMPP/MySQL).
- Reforzar conocimientos de desarrollo web con Java y Spring Boot.
- Mejorar la comprensión y práctica de seguridad en aplicaciones web con Spring Security.
- Trabajar en equipo aplicando buenas prácticas de colaboración en proyectos con GitHub.
- Desarrollar capacidades de planificación y gestión de tiempos en un proyecto académico realista.

#### 1.4 Planificación del trabajo

La planificación se organizó en tres sprints de cuatro semanas cada uno, siguiendo la metodología Scrum:

Fase / Sprint	Actividades principales	Duración
Sprint 1 – Análisis y Diseño	Levantamiento de requisitos, definición de casos de uso, diseño de la base de datos y estructura del proyecto en Spring Boot.	Semanas 1 - 4
Sprint 2 – Backend y Seguridad	Desarrollo de la API REST, configuración de la conexión a MySQL, creación de entidades, repositorios y servicios; integración de Spring Security con JWT	Semanas 5 - 9
Sprint 3 – Frontend e Integración	Maquetación con Thymeleaf, desarrollo de las vistas (login, catálogo, detalles de películas, administración), consumo de la API y pruebas funcionales.	Semanas 9 - 13
Cierre	Pruebas finales, documentación, corrección de errores y preparación de la presentación.	Semana 13 - 18

## 2 Estado del Arte

### 2.1 Tecnologías

Este proyecto se ha construido sobre una base de diferentes tecnologías para cada función importante del sistema o lógica de la página Streams UTP

A continuación, se detallarán las distintas tecnologías consideradas para cada parte del desarrollo.

#### 2.1.1. Tecnologías para el desarrollo de APIs REST

##### 2.1.1.1. Concepto de un API REST

Un API REST es una forma de construir servicios web que usan “reglas” (REST) que en realidad comparten principios básicos, como buenas prácticas para que distintas aplicaciones puedan intercambiar información de manera rápida y ordenada, igual que si siguieran un mismo idioma para comunicarse.

En este proyecto se construirá una API RESTful que reciba solicitudes desde un frontEnd, procese esas solicitudes accediendo a una base de datos para obtener o modificar la información requerida y, finalmente, responda al frontEnd enviando los datos solicitados en el formato correspondiente.

**Cliente para API de Películas (Demo)**

**Autenticación (Usuario ADMIN):**

Usuario: adminLOL  
Contraseña: [Tu\_Contraseña\_Secreta]

1. Crear Nueva Película (Ejemplo)    2. Obtener Todas las Películas

ID de Película para GET/DELETE:     3. Obtener Película por ID    4. Eliminar Película por ID

**API Response:**

```
Todas las Películas:
[
  {
    "id": 1,
    "titulo": "IronMan 3",
    "imagen": "/imagenes/cuadricula/1.jpg",
    "precioComprar": 19.99,
    "precioAlquilar": 5.99
  },
  {
    "id": 2,
    "titulo": "Thor: Love and Thunder",
    "imagen": "/imagenes/cuadricula/2.jpg",
    "precioComprar": 17.99,
    "precioAlquilar": 5.99
  },
  {
    "id": 3,
    "titulo": "Thor: Ragnarok",
    "imagen": "/imagenes/cuadricula/3.jpg",
    "precioComprar": 14.99,
    "precioAlquilar": 4.99
  }
]
```

Figura 2: Vista de la api response

## 2.1.2. Tecnologías para el desarrollo del FrontEnd y BackEnd

### 2.1.2.1. Frontend (HTML, CSS, Bootstrap, Thymeleaf)

Es la parte visual del software, desde el punto de vista del usuario, en nuestra página usamos tecnologías como el uso de Bootstrap para diseños de la página, Thymeleaf para generar plantillas. Todo eso aplicado en el HTML.



Figura 3: Framework Bootstrap y motor de plantillas Thymeleaf

### 2.1.2.2. Backend (Java Maven, JSP, Sprint Boot)

En todo lo que conlleva al núcleo de la página se basa en la aplicación del framework Spring más específico con la Herramienta de Spring Boot. y como herramienta de construcción, se construye y gestiona con Maven.

En el Caso de Seguridad de la Pagina se implementa Spring Security para gestionar la autenticación de usuarios y el control de acceso mediante por ejemplo validaciones, encrypciones, etc.

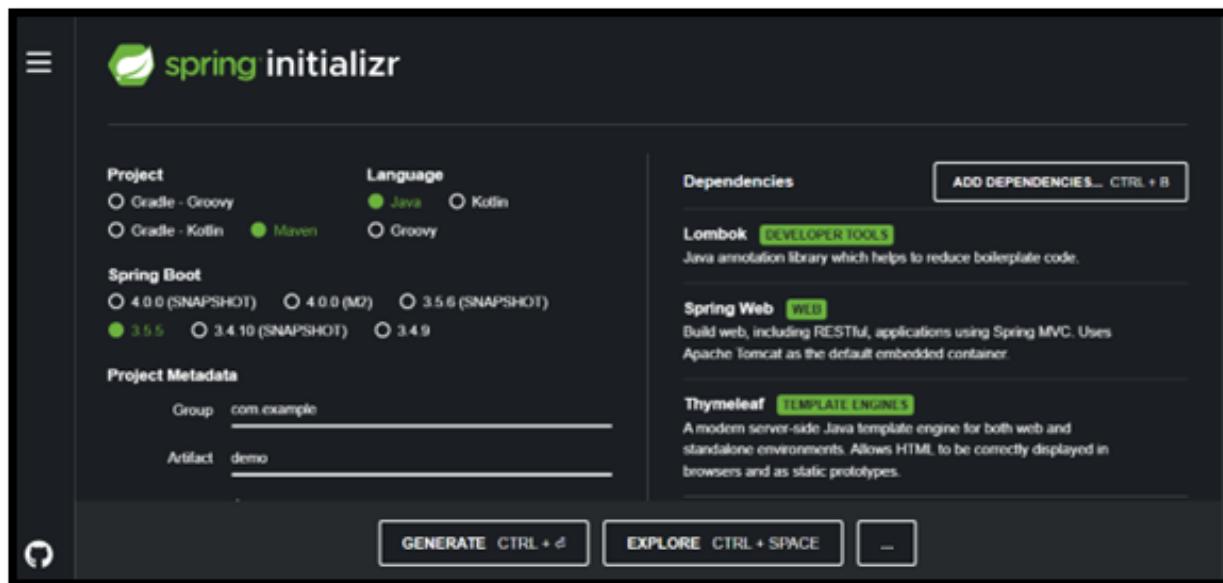


Figura 4: Initializr de Spring Boot

### 2.1.2.3. Base de Datos (MySQL)

Con el uso de la herramienta Spring Boot, aplicamos Spring Data JPA que conecta a una base de datos MySQL

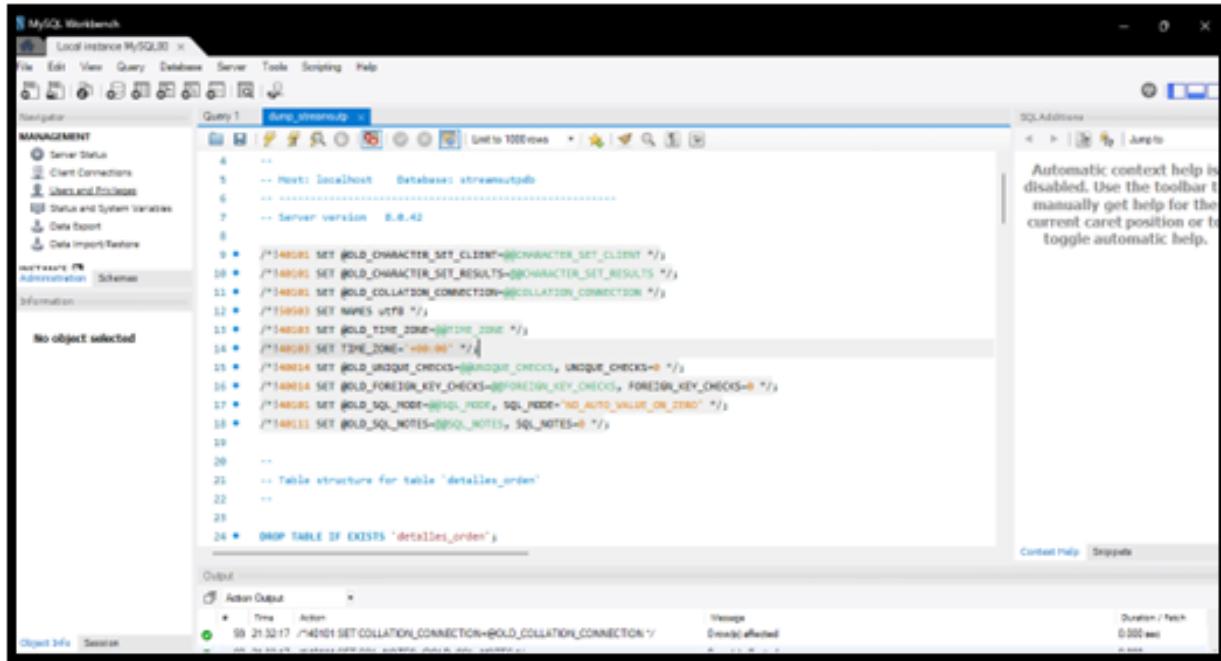


Figura 5: Vista de la base de datos MySQL

### 2.1.3. Tecnologías para la extracción automática de información

Actualmente, el proyecto ya aplica una forma de extracción automática de información a través de Spring Data JPA y Hibernate. Esta tecnología extrae automáticamente los datos de la base de datos MySQL y los mapea a objetos Java, eliminando la necesidad de escribir consultas SQL manualmente.

```
@Entity // 1. Le dices a Hibernate: "Esta clase representa una tabla en la BD".
@Table(name = "peliculas") // 2. Le dices: "La tabla se llama 'peliculas'".
public class Pelicula {

    @Id // 3. Le dices: "Este campo es la clave primaria (el identificador único)".
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    // 4. Hibernate asume que este campo 'titulo' corresponde a una columna llamada 'titulo'.
    private String titulo;

    private String descripcion;

    private String imagen;

    @Column(name = "precio_comprar") // 5. Si el nombre de la columna es diferente al del campo, lo especificas.
    private Double precioComprar;
```

Figura 6: clase pelicula donde estaran los datos

#### 2.1.4. Otras tecnologías

- **Node js:** Se pensó aplicar JavaScript dentro de un entorno de prueba Typescript, podría ser una opción viable.
- **Angular / React:** Dos opciones de Framework de Frontend para diseño de apps / páginas webs
- **H2 DATABASE:** Es una base de datos en base de Spring que funciona en memoria.

## 2.2 Fuentes de datos

El estudio de las fuentes de datos para StreamsUTP se dividió en dos vertientes: un análisis de las fuentes de información necesarias para la carga inicial de películas en la base de datos y una revisión de plataformas similares que ofrecen servicios de streaming o alquiler de contenido digital, a fin de identificar buenas prácticas y necesidades del mercado.

### 2.2.1. Fuentes de información

Para poblar la base de datos de películas, se requieren fuentes con información estructurada, estandarizada y completa, que incluyan títulos, géneros, sinopsis, reparto, portadas y valoraciones. Por lo que, algunas de las principales fuentes contempladas son:

- IMDb (<https://www.imdb.com/>): plataforma internacional ampliamente reconocida que provee datos completos y confiables sobre películas, actores, géneros y valoraciones de usuarios.

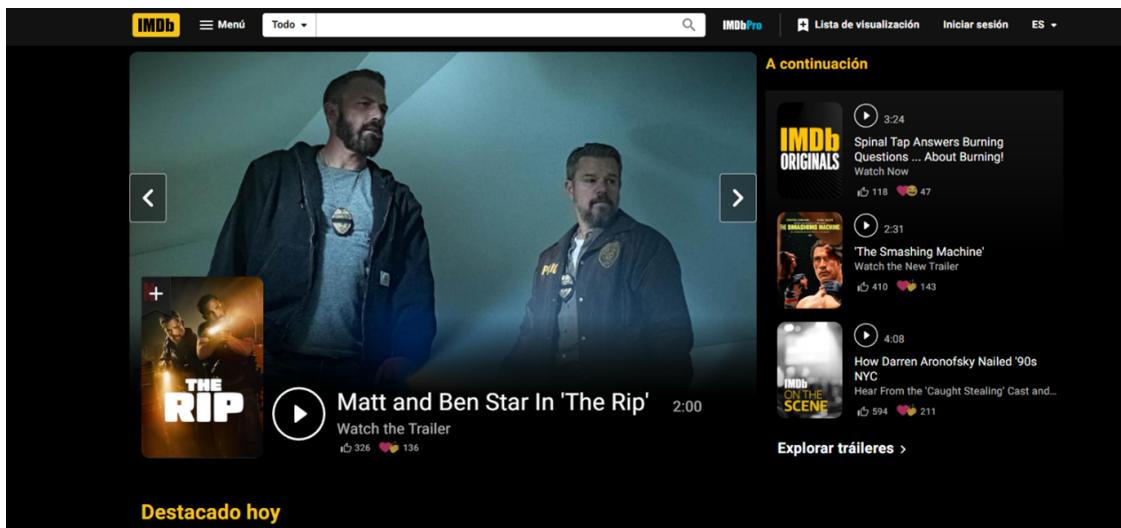


Figura 7: Vista de la aplicación de la fuente 1

- The Movie Database – TMDb (<https://www.themoviedb.org/base>) de datos colaborativa que ofrece una API pública gratuita, con información en múltiples idiomas, ideal para integraciones automáticas.

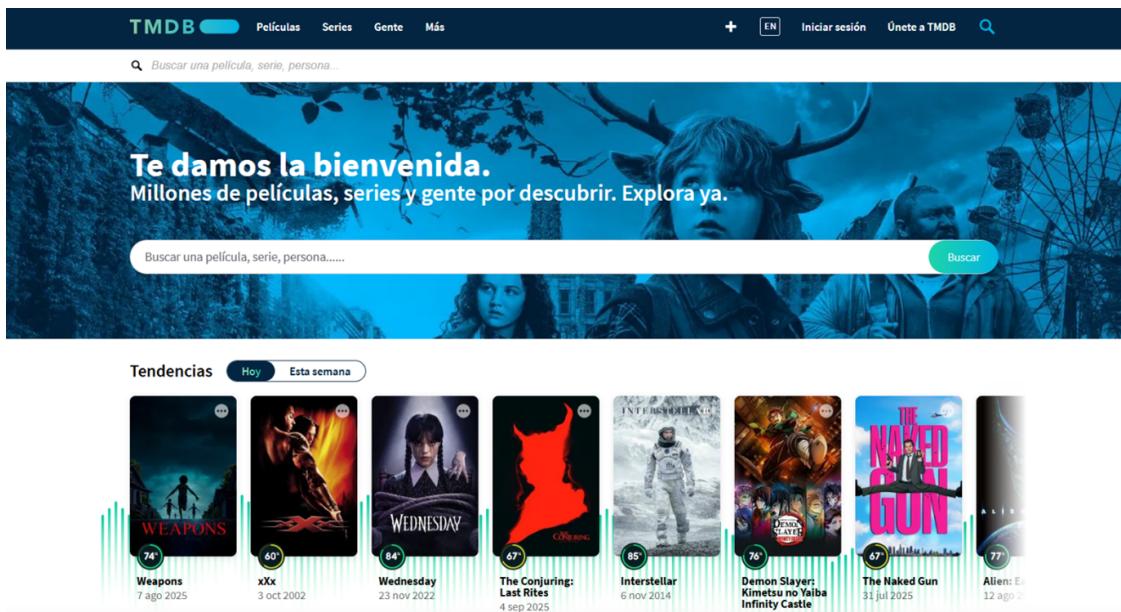


Figura 8: Vista de la aplicación de la fuente 2

Estas fuentes proporcionan la base inicial para poblar el sistema y garantizar que los

catálogos de películas estén actualizados y organizados.

### 2.2.2. Webs dedicadas al streaming de películas

- **Netflix:** Plataforma líder en suscripción de streaming. Referente para interfaz de usuario intuitiva y sistema de navegación por categorías.
- **Amazon Prime Video:** Combina suscripción con compra/alquiler individual. Presenta estructura clara de precios diferenciados y gestión de biblioteca personal.
- **Google TV:** Ofrece compra y alquiler de películas individuales, con interfaz clara que facilita la selección de tipo de transacción (comprar vs alquilar). Excelente modelo para el diseño de carrito y confirmación de órdenes.

El análisis de estas plataformas permitió identificar funcionalidades clave que los usuarios esperan: navegación intuitiva del catálogo, diferenciación clara entre opciones de compra y alquiler, historial de órdenes y un sistema seguro de gestión de pagos.

### 2.2.3. Análisis de lo estudiado

Tras el análisis de las fuentes, se concluye que, si bien existen catálogos digitales de películas con información extensa, no todos presentan datos estandarizados que permitan su integración directa de forma automática. Sin embargo, plataformas como TMDb destacan por ofrecer una API abierta y organizada que facilita la carga inicial del catálogo en StreamsUTP. Por otro lado, las plataformas de streaming consolidadas (Netflix, Amazon Prime Video, Google TV) sirven como referencia en cuanto a usabilidad y funcionalidades, destacando la navegación intuitiva, la claridad en las opciones de compra y alquiler, y la gestión de pedidos. En este sentido, StreamsUTP cumple un papel innovador al unificar en una sola aplicación la exploración, compra y alquiler de películas, junto con la administración de

usuarios y órdenes, brindando una plataforma estandarizada, clara y práctica para clientes y administradores.

### 3 Desarrollo

#### 3.1 Diseño de la base de datos

La base de datos se ha intentado simplificar lo máximo posible, ofreciendo una solución sólida que evite complicaciones innecesarias. El diagrama que la representa es el siguiente:

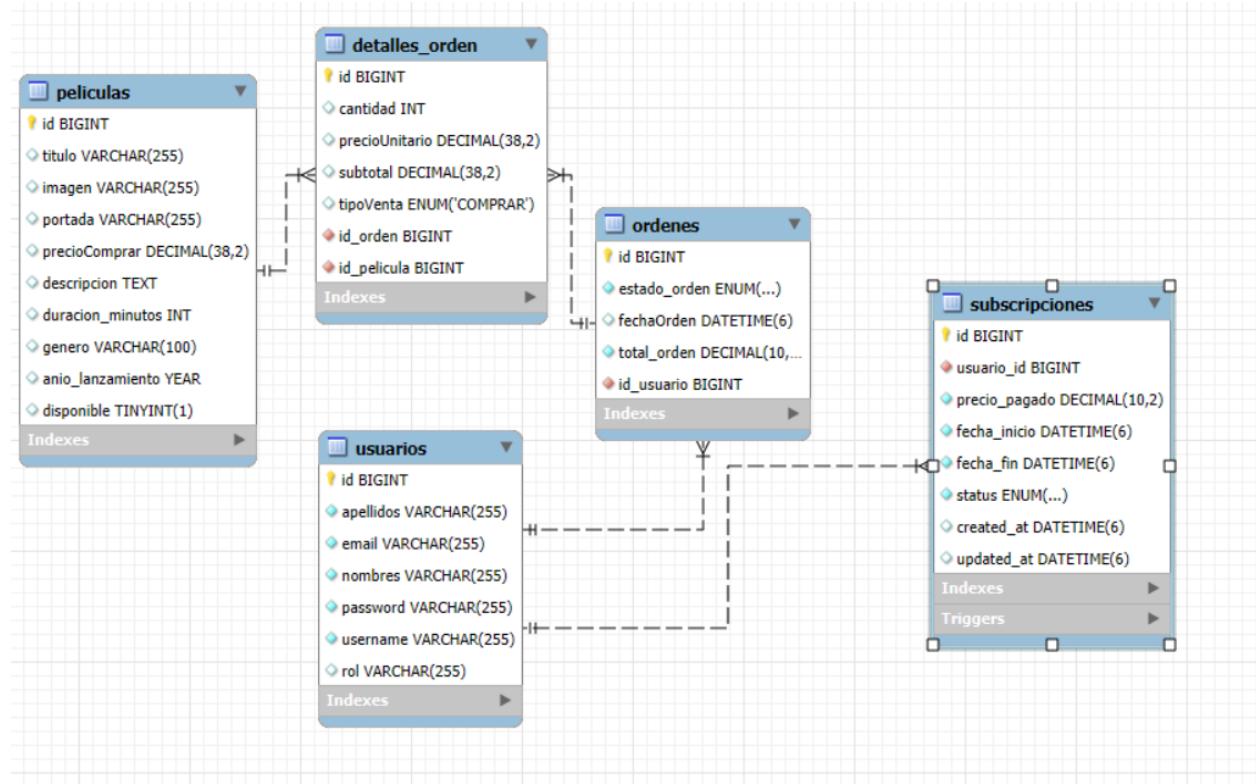


Figura 9: Diagrama de la Base de Datos

### 3.2 BackEnd

Los EndPoints desarrollados serán los siguientes

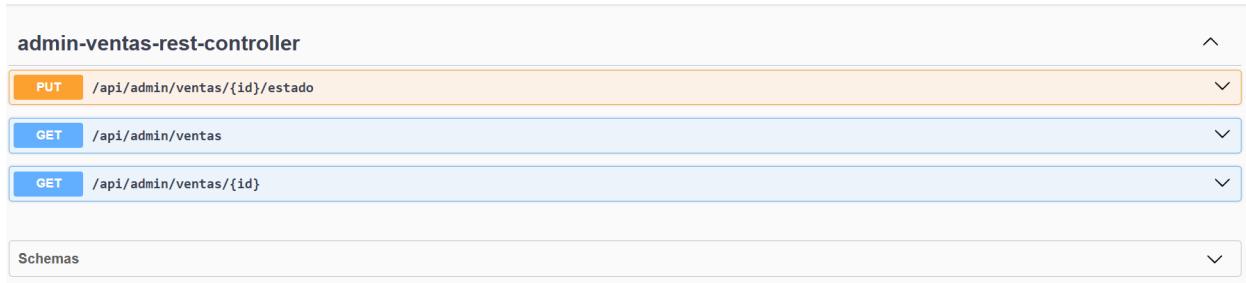


Figura 10: EndPoints del BackEnd



Figura 11: Diagrama de la Base de Datos

A continuación, se mostrará la estructura de clases y paquetes y se explicará la función de cada uno:



Figura 12: Diagrama de la Base de Datos

### 3.2.1. Entidades sencillas

Las entidades sencillas del sistema son aquellas con estructuras simples, operaciones básicas y relaciones mínimas. En este proyecto se identifican tres entidades sencillas: **EstadoOrden**, **TipoVenta** y **CarritoItem**.

- **EstadoOrden:** es un enumerado que define los cuatro posibles estados de una orden en el sistema(PENDIENTE, PROCESADA, COMPLETADA y CANCELADA) garantizando que sólo puedan asignarse estos valores y evitando estados inválidos; se emplea en la entidad Orden con @Enumerated(EnumType.STRING) para persistir el estado como texto en la base de datos y centraliza la lógica de flujo de la transacción, facilitando la gestión y futura extensión de los estados.
- **TipoVenta:** es un enumerado con un único valor COMPRAR, que representa el tipo de transacción permitido en el sistema; al usar un enum se garantiza la integridad de

los posibles tipos de venta y se facilita su ampliación futura.

- **CarritoItem** es un DTO serializable que modela un elemento temporal del carrito de compras, con los atributos esenciales para la sesión: peliculaId (identificador de la película), titulo, tipo (por ejemplo, ALQUILER o COMPRA), precio (como BigDecimal para precisión monetaria), imagen y cantidad; al no ser una entidad JPA, su ciclo de vida se maneja en memoria mediante la sesión HTTP y se utiliza para transferir datos entre la vista y el servicio de carrito.
- **CarritoController** gestiona todas las operaciones relacionadas con el carrito de compras, incluyendo la adición, visualización, eliminación de ítems y el procesamiento final de la compra. Utiliza la sesión HTTP para almacenar temporalmente los CarritoItem y coordina con los servicios de Orden, Pelicula y Usuario para validar datos y persistir la transacción.
- **agregarAlCarrito/CarritoController** (@PostMapping("/agregar")): recibe los parámetros de la película (ID, título, tipo, imagen, cantidad), valida que el tipo sea “comprar”, obtiene el precio real de la entidad Pelicula, crea un nuevo CarritoItem y lo añade a la sesión; proporciona mensajes de éxito o error mediante RedirectAttributes.
- **mostrarCarrito/CarritoController** (@GetMapping()): verifica que el usuario esté autenticado, recupera la lista de CarritoItem de la sesión, calcula el total sumando los precios y añade ambos al modelo para renderizar la vista del carrito.
- **eliminarDelCarrito/CarritoController** (@PostMapping("/eliminar")): elimina un elemento del carrito según su índice en la lista de sesión, actualiza la sesión y genera un mensaje de confirmación o error.
- **procesarCompra/CarritoController** (@PostMapping("/procesar-compra")): comprueba la autenticación, obtiene el usuario logueado, valida que el carrito no esté vacío,

invoca ordenService.crearOrden() para persistir la orden con todos los ítems y limpia la sesión tras una compra exitosa; maneja excepciones con logging y mensajes de error.

- **RegistroController:** Controlador encargado de manejar el registro de nuevos usuarios. Expone la ruta /registro para mostrar el formulario de registro (mostrarFormulario) y procesa los datos enviados mediante POST (procesarFormulario). En este último método valida que las contraseñas coincidan y que no existan errores de validación. También verifica que el usuario o email no estén ya registrados mediante el servicio de usuarios (usuarioService). Si el registro es exitoso, redirige a la página de ingreso con un mensaje de bienvenida.
- **PerfilController:** Controlador que permite a los usuarios autenticados acceder a su perfil personal mediante la ruta /perfil. Obtiene el nombre de usuario autenticado a través del objeto Principal de Spring Security y consulta el servicio de usuarios para recuperar los datos correspondientes. En caso de que el usuario no se encuentre (evento raro si está autenticado), redirige a la página de ingreso con un mensaje de error. Muestra los datos del usuario en la vista perfil.
- **IngresoController:** Controlador que maneja la presentación del formulario de inicio de sesión en la ruta /ingreso. Utiliza parámetros opcionales para mostrar mensajes en la interfaz cuando ocurre un error en el inicio de sesión (como credenciales incorrectas) o cuando un usuario ha cerrado sesión exitosamente. No contiene lógica de autenticación directa, dado que Spring Security se encarga de ella.
- **InicioController:** Controlador que gestiona la página principal del sistema, accesible desde las rutas / o /index. Recupera el usuario autenticado, si existe, y lo añade al modelo para personalizar la vista junto con una lista de todas las películas disponibles consultadas desde el repositorio. Además, expone la ruta /confirmacion-compra para mostrar la pantalla de confirmación tras una compra exitosa. Facilita la interacción inicial y la presentación de contenidos públicos o personalizados.

### 3.2.2. Entidades Complejas

- **Usuario:** La entidad representa a cada usuario registrado en el sistema, almacenando información personal como apellidos, nombres, email, username, contraseña y rol. Implementa la interfaz UserDetails de Spring Security, lo que permite su integración directa en los procesos de autenticación y autorización. Además, mantiene relaciones uno-a-muchos con las entidades Orden y Subscripcion, permitiendo que cada usuario tenga varias órdenes y suscripciones asociadas. Entre sus métodos principales destacan getAuthorities(), que devuelve los roles del usuario para la autorización; los métodos de estado de cuenta (isAccountNonExpired(), isAccountNonLocked(), isCredentialsNonExpired(), isEnabled()), que siempre retornan verdadero; y los métodos utilitarios addOrden(Orden orden) y addSubscripcion(Subscripcion subscripcion), que facilitan la gestión bidireccional de las relaciones con órdenes y suscripciones.
- **UserRepository:** La interfaz extiende JpaRepository<Usuario, Long> y permite realizar operaciones CRUD sobre la entidad de usuarios en la base de datos. Incluye métodos personalizados como findByUsername y findByEmail para buscar usuarios por nombre de usuario o correo electrónico, así como findUsersWithActiveSubscription para obtener usuarios con suscripción activa. Esta interfaz es utilizada por el servicio de usuarios para acceder y manipular los datos persistidos.
- **UserService:** El servicio implementa la lógica de negocio relacionada con los usuarios. Gestiona el registro de nuevos usuarios, la autenticación, la actualización y eliminación de registros, la búsqueda por distintos criterios y la encriptación de contraseñas. Utiliza la interfaz UserRepository para interactuar con la base de datos y expone métodos como loadUserByUsername para la autenticación, registrar para el registro de usuarios, findByUsername, findUsuarioByEmail y findUsuarioById para búsquedas, findAllUsuarios para obtener todos los usuarios, saveUsuario para guardar o actuali-

zar registros, deleteUsuario para eliminar usuarios y findSubscribedUsers para obtener usuarios con suscripción activa.

- **UsuarioOrdenController:** El controlador expone endpoints que permiten a los usuarios autenticados consultar su historial de órdenes y los detalles de cada orden. Utiliza el servicio de usuarios para obtener la información del usuario autenticado y el servicio de órdenes para recuperar las órdenes asociadas. Sus métodos principales son verHistorialOrdenes, que muestra el historial de órdenes del usuario autenticado, y verDetalleOrdenUsuario, que muestra el detalle de una orden específica validando que pertenezca al usuario. El controlador se encarga de validar permisos y mostrar mensajes de error cuando corresponde, asegurando la correcta visualización y protección de la información.
- **Pelicula:** Entidad que representa las películas disponibles en el sistema. Contiene atributos como id, titulo, imagen, portada, precioComprar, descripcion, duracionMinutos, genero, anioLanzamiento y un indicador booleano disponible para controlar su disponibilidad. Mantiene una relación uno-a-muchos con DetalleOrden, que representa las órdenes en las que la película ha sido incluida. Esta entidad es fundamental para el catálogo de productos y su gestión.
- **PeliculaService:** Servicio que encapsula la lógica de negocio relacionada con las películas. Proporciona métodos para guardar o actualizar películas (guardarPelicula), obtener todas las películas (obtenerTodasLasPelículas), buscar una película por su ID (obtenerPeliculaPorId) y eliminar una película (eliminarPelicula). Este servicio utiliza internamente el repositorio para acceder a la base de datos y puede extenderse con métodos adicionales para búsquedas personalizadas o lógica específica.
- **PeliculaPublicController:** Controlador que expone endpoints públicos para que los usuarios puedan consultar detalles de una película específica. Su método principal verDetalle recibe el ID de la película, consulta el servicio para obtener la entidad y, si

existe, la añade al modelo para mostrar la vista de detalle; en caso contrario, redirige a la página principal.

- **PeliculaController:** Controlador administrativo protegido por seguridad que permite a los usuarios con rol ADMIN gestionar el catálogo de películas. Incluye métodos para listar todas las películas (listarPelículas), mostrar formularios para crear (mostrarFormularioNuevo) o editar (mostrarFormularioEditar) películas, guardar cambios (guardarPelícula) y eliminar películas (eliminarPelícula). Utiliza validaciones y redirectiones con mensajes flash para informar al usuario sobre el resultado de las operaciones.
- **Orden:** La entidad representa una transacción de compra realizada por un usuario en el sistema. Almacena información como el usuario que realizó la orden, la fecha de la transacción, el total de la orden, el estado actual (usando el enum EstadoOrden) y una lista de detalles (DetalleOrden) que especifican los productos adquiridos. Incluye métodos para inicializar la orden con valores por defecto y para añadir detalles, manteniendo la relación bidireccional con los elementos comprados.
- **OrdenService:** El servicio encapsula la lógica de negocio relacionada con las órdenes. Permite crear una nueva orden a partir de los ítems del carrito (crearOrden), eliminarlas (deleteOrden), consultar todas las órdenes (findAllOrdenes), buscar una orden por su ID (findOrdenById), actualizar el estado de una orden (actualizarEstadoOrden) y obtener todas las órdenes asociadas a un usuario específico (findOrdenesByUsuario). Este servicio coordina con los servicios y repositorios de usuario, película y detalle de orden para validar datos y persistir la información correctamente.
- **OrdenRepository:** La interfaz extiende JpaRepository<Orden, Long> y permite realizar operaciones CRUD sobre las órdenes en la base de datos. Incluye métodos personalizados para buscar órdenes por usuario (findByUsuario) y por estado (findByEstadoOrden), facilitando la consulta de transacciones según distintos criterios. Esta

interfaz es utilizada por el servicio de órdenes para acceder y manipular los datos persistidos.

- **Subscripcion:** La entidad representa la suscripción de un usuario al sistema, almacenando información como el precio pagado, las fechas de inicio y fin, el estado actual (activo, expirado o cancelado) y las marcas de tiempo de creación y actualización. Mantiene una relación muchos-a-uno con la entidad Usuario, permitiendo que cada usuario tenga varias suscripciones asociadas. Incluye métodos para verificar si la suscripción está activa o ha expirado, y utiliza anotaciones para gestionar automáticamente las fechas de creación y actualización.
- **SubscripcionService:** El servicio encapsula la lógica de negocio relacionada con las suscripciones. Permite crear nuevas suscripciones (crearSubscripcion), buscar todas las suscripciones de un usuario (findByUsuario), obtener la suscripción activa (findActiveSubscription), listar todas las suscripciones (findAll) y cancelar una suscripción (cancelarSubscripcion). Este servicio utiliza el repositorio para acceder y manipular los datos persistidos, y se encarga de validar y actualizar el estado de las suscripciones.
- **SubscripcionRepository:** La interfaz extiende JpaRepository<Subscripcion, Long> y permite realizar operaciones CRUD sobre las suscripciones en la base de datos. Incluye métodos personalizados para buscar suscripciones por usuario (findByUsuario) y para obtener la suscripción activa de un usuario (findByUsuarioAndStatus). Esta interfaz es utilizada por el servicio de suscripciones para facilitar la consulta y gestión de las suscripciones.
- **SubscripcionController:** El controlador expone endpoints para que los usuarios puedan gestionar sus suscripciones, incluyendo la creación de nuevas suscripciones, la visualización del historial y la cancelación de suscripciones (esta última solo accesible para administradores). Utiliza el servicio de suscripciones para procesar las operaciones y el servicio de usuarios para obtener la información del usuario autenticado. Sus

métodos principales permiten mostrar el formulario de suscripción, procesar la creación, listar el historial y cancelar suscripciones, gestionando la navegación y los mensajes de confirmación o error.

- **AdminUsuarioController:** Controlador administrativo que gestiona las operaciones relacionadas con usuarios del sistema. Proporciona métodos para listar todos los usuarios (listarUsuarios), listar usuarios con suscripciones activas (listarUsuariosSuscritos), mostrar formularios para crear nuevos usuarios (mostrarFormularioNuevoUsuario) y editar usuarios existentes (mostrarFormularioEditarUsuario). Permite además guardar usuarios nuevos o actualizados (guardarUsuario) y eliminar usuarios (eliminarUsuario). Este controlador utiliza UsuarioService para acceder y modificar los datos, y maneja redirecciones y mensajes flash para informar al administrador sobre el éxito o fallo de las operaciones.
- **AdminVentasController:** Controlador que administra las órdenes de venta del sistema, accesible únicamente para usuarios con rol ADMIN. Permite listar todas las órdenes (listarOrdenes), ver el detalle específico de una orden (verDetalleOrden) y mostrar los estados disponibles para cambio. Ofrece la posibilidad de actualizar el estado de una orden (actualizarEstadoOrden) y eliminar órdenes (eliminarOrden), manejando errores y confirmaciones mediante mensajes flash. Este controlador utiliza OrdenService para la lógica de negocio y asegura la integridad y seguridad mediante la anotación @PreAuthorize.

### 3.3 Seguridad de la API

La configuración de seguridad está definida en la clase SecurityConfig, que habilita y personaliza la protección de las rutas en la aplicación utilizando Spring Security.

Se configura un WebSecurityCustomizer para ignorar los recursos estáticos como CSS,

JS, imágenes y favicon, permitiendo su acceso libre sin autenticación.

El SecurityFilterChain define las reglas principales de autorización:

- Las rutas /ingreso, /registro y la raíz / están abiertas a todos, sin necesidad de autenticación.
- Las rutas que empiezan con /carrito/\*\* requieren que el usuario esté autenticado para acceder, asegurando que sólo usuarios con sesión válida puedan interactuar con el carrito.
- Las rutas bajo /admin/\*\* están protegidas con el rol ADMIN, restringiendo el acceso solo a administradores.
- Cualquier otra ruta también requiere autenticación, evitando accesos anónimos no autorizados.

El mecanismo de login utiliza una página personalizada en /ingreso y redirige siempre al usuario autenticado a la página principal / tras un inicio exitoso.

El logout es accesible para cualquier usuario autenticado o no, permitiendo cerrar sesión libremente.

Para la encriptación de contraseñas se utiliza el PasswordEncoder de tipo BCryptPasswordEncoder, considerado seguro para almacenar credenciales.

La configuración actual incluye un comentario sugerente sobre la desactivación temporal de CSRF para pruebas, recordando que en producción debe habilitarse y manejarse adecuadamente para proteger contra ataques de tipo CSRF.

Esta configuración asegura un control fino de acceso y una integración fluida con las funcionalidades de login, logout y roles personalizados en la aplicación.

### 3.4 FrontEnd

A continuación, se mostrarán las distintas vistas presentes en el FrontEnd de la aplicación de manera similar a la navegación esperada por los usuarios.

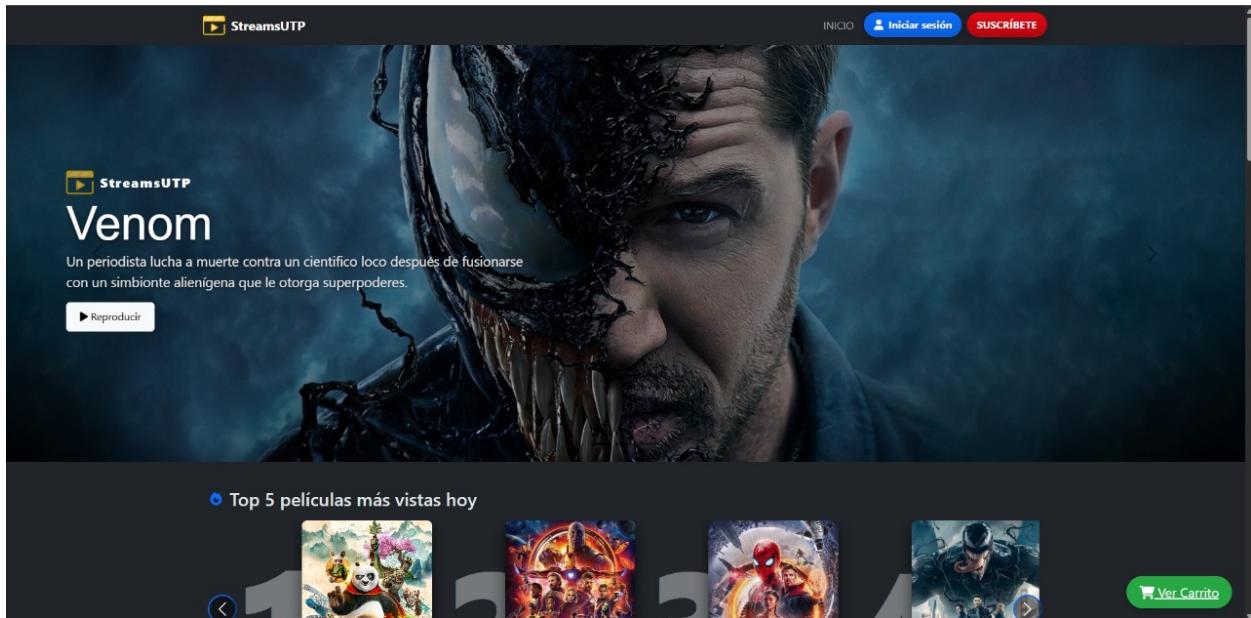


Figura 13: FrontEnd del proyecto

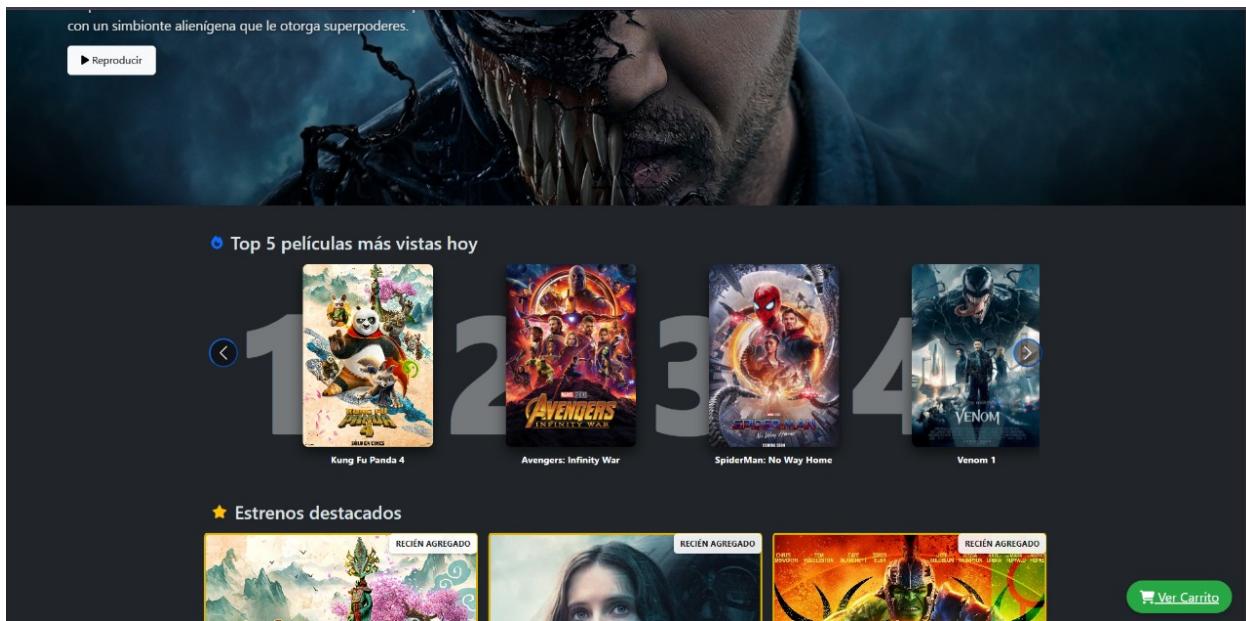


Figura 14: FrontEnd del proyecto

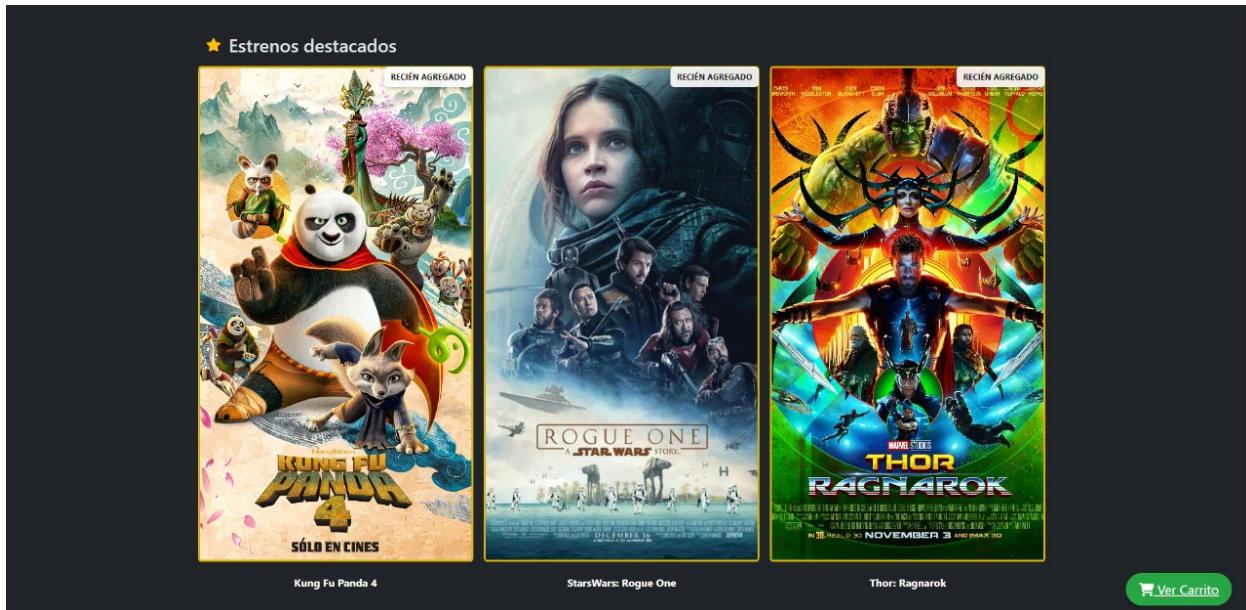


Figura 15: FrontEnd del proyecto

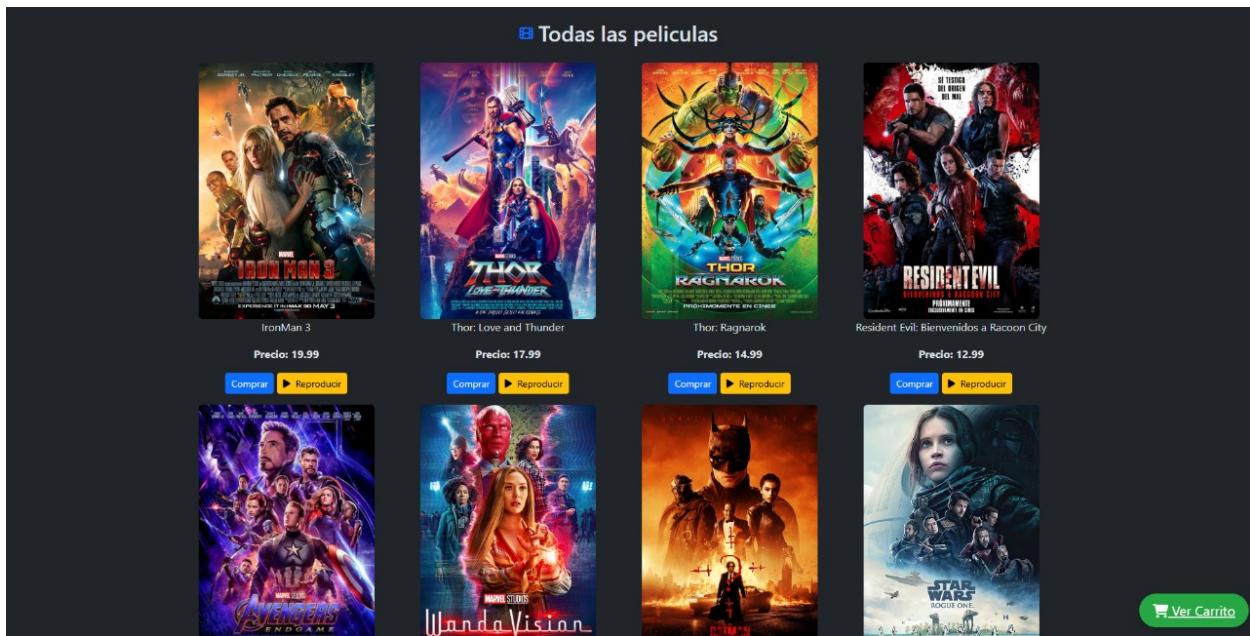


Figura 16: FrontEnd del proyecto

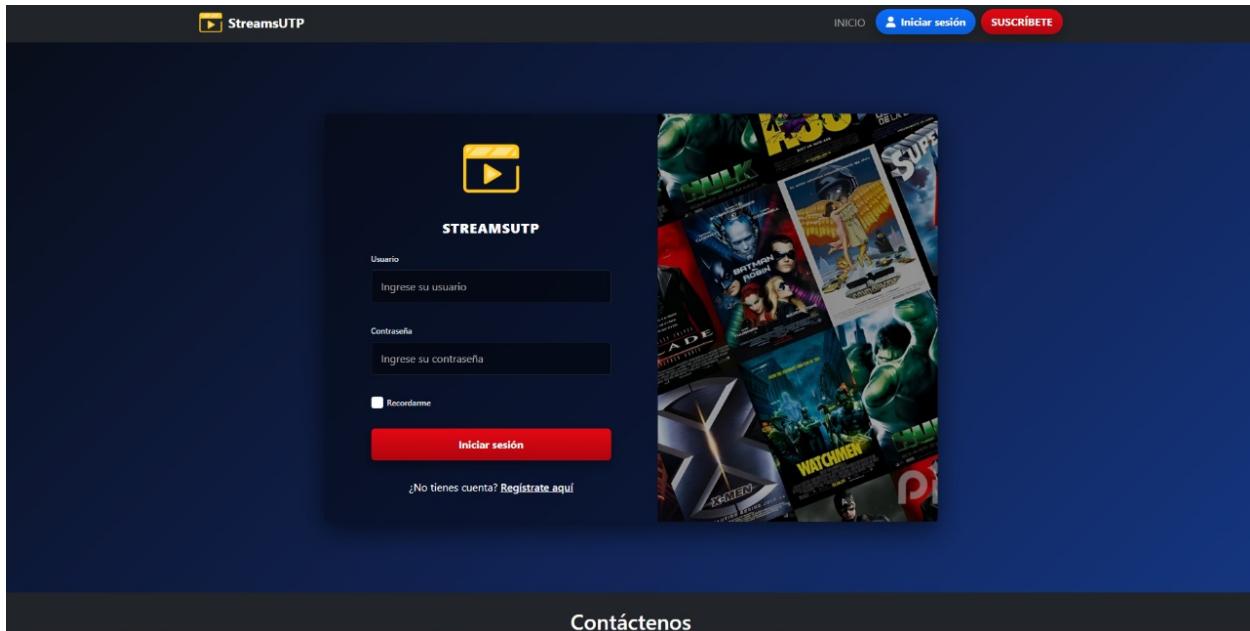


Figura 17: FrontEnd del proyecto

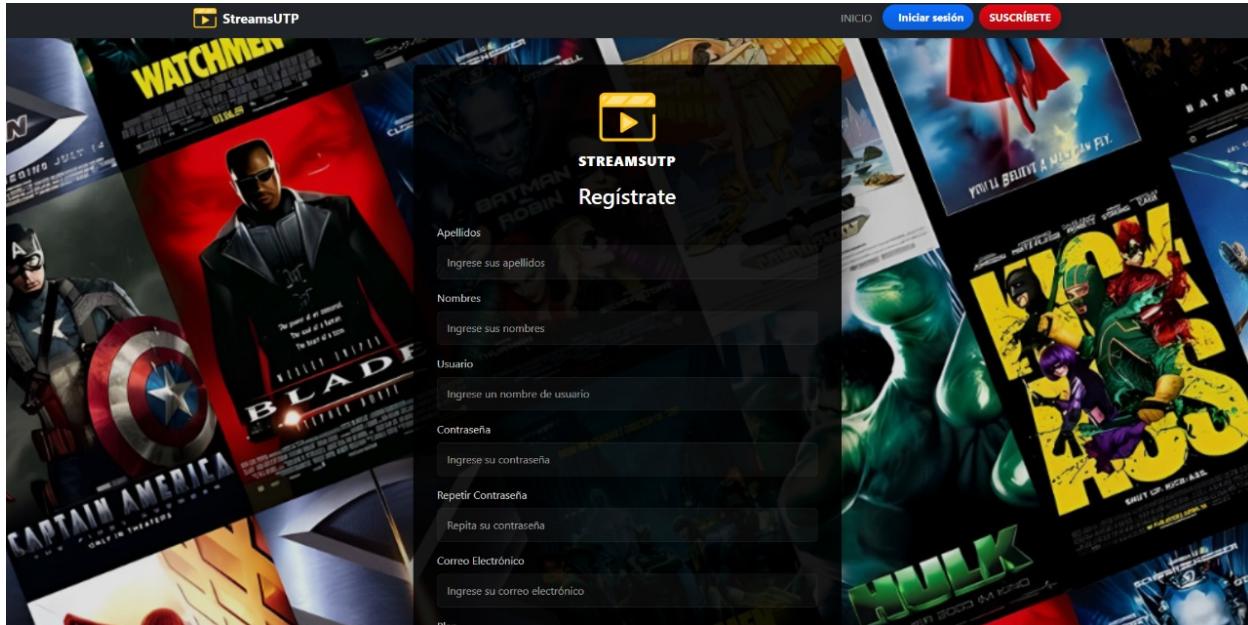


Figura 18: FrontEnd del proyecto

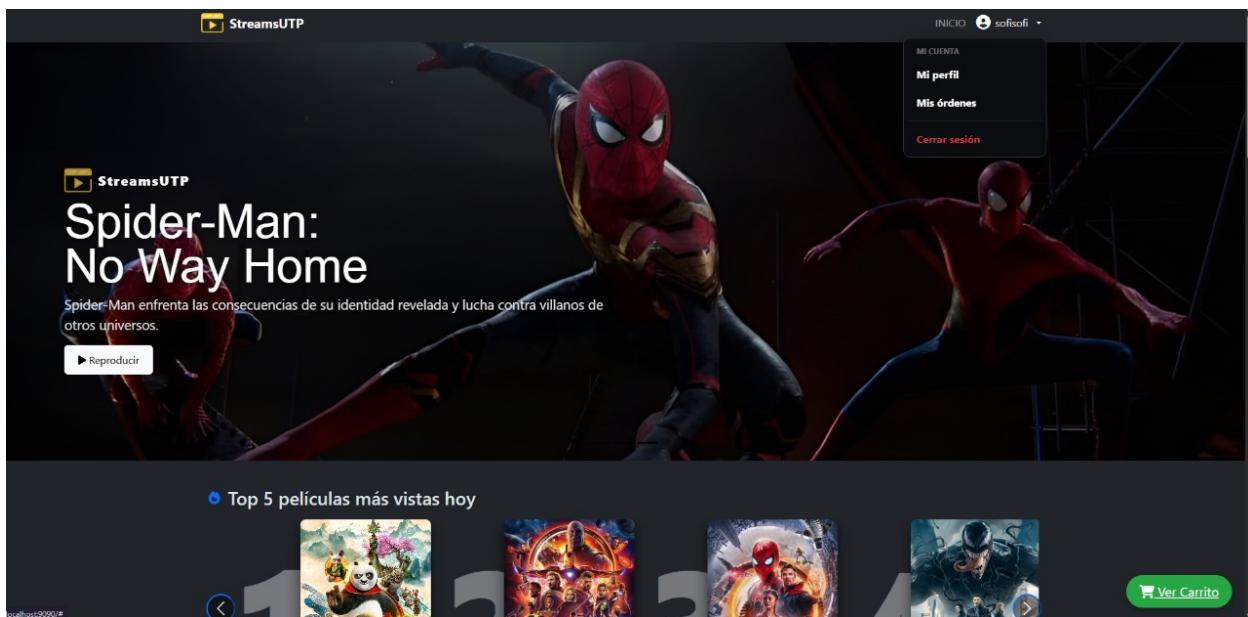


Figura 19: FrontEnd del proyecto

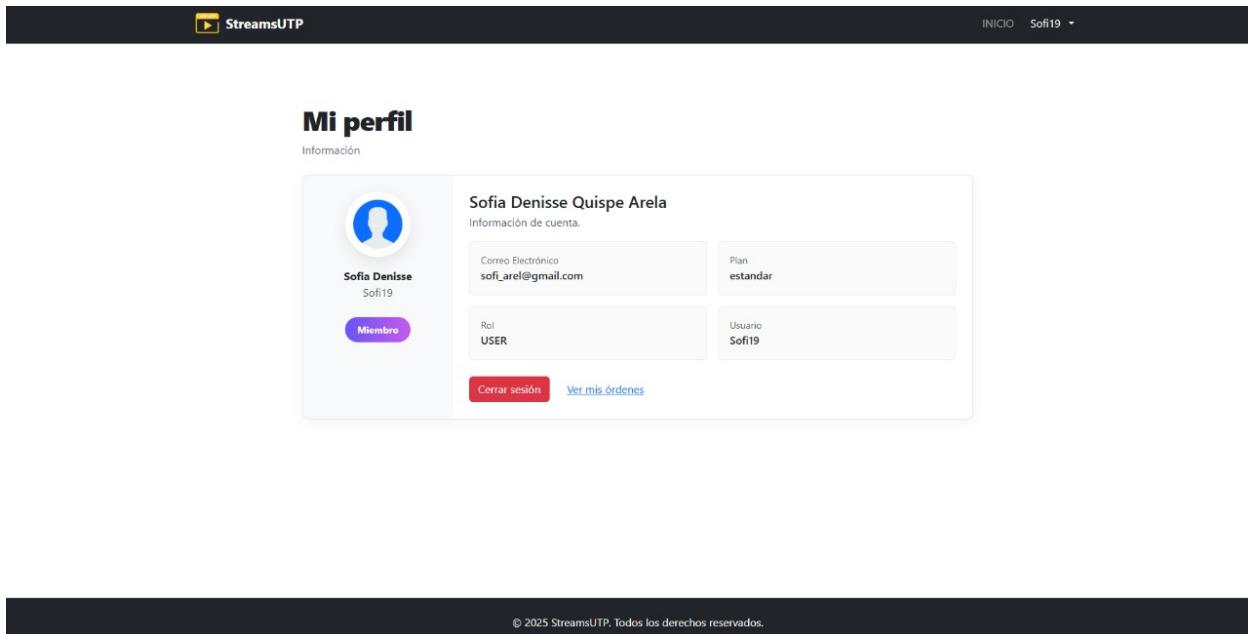


Figura 20: FrontEnd del proyecto