INSTITUTO POLITÉCNICO NACIONAL

“Escuela Superior de Cómputo”

PRÁCTICA 3

”Modelo De Análisis Y Consumo De Apis En Proyectos De Software”

MATERIA: INGENIERÍA DE SOFTWARE

PROFESOR: Gabriel Hurtado Avilés

GRUPO: 6CV3

INTEGRANTES: -Almogabar Nolasco Jaime Brayan | 2022630476

-Díaz Hernández Braulio | 2022630489

-García Quiroz Gustavo Ivan | 2022630278

- Morales Torres Alejandro | 2021630480

-Rodriguez Rivera Claudia Patricia | 2022630334

-Tellez Partida Mario Iahveh | 2022630535

Índice

1 Introducción

La Práctica 3 de Ingeniería de Software tiene como propósito fundamental desarrollar un modelo de análisis detallado para un proyecto de software, integrando diversos componentes técnicos y metodológicos. El objetivo principal es crear una documentación completa que represente la estructura, funcionalidades y características del sistema mediante la utilización de herramientas de modelado UML (Lenguaje de Modelado Unificado).

En esta práctica, se busca profundizar en la comprensión y documentación de un proyecto de software, más allá de su implementación básica. Se requiere un análisis exhaustivo que incluya múltiples diagramas y documentaciones que permitan una visión integral del sistema, abordando aspectos como la estructura de datos, interacciones de usuarios, flujos de información y diseño conceptual.

Adicionalmente, la práctica plantea la integración de servicios REST y el consumo de APIs públicas, lo que añade un componente práctico de conexión e interoperabilidad al proyecto. Los estudiantes deben demostrar su capacidad para conectar sus sistemas con servicios externos, procesar datos en formatos como JSON o XML, e implementar funcionalidades que amplíen las capacidades originales del software.

Los principales elementos a desarrollar incluyen:

- Documentación detallada mediante diagramas UML
- Modelo de análisis completo del sistema
- Integración de APIs públicas
- Implementación de funcionalidades adicionales
- Documentación bajo el modelo FURPS+

El propósito final es formar profesionales capaces de diseñar, documentar y expandir sistemas de software de manera sistemática y profesional, integrando herramientas de modelado, análisis de requisitos y tecnologías de integración de servicios.

2 Objetivos de la práctica

El objetivo de esta práctica es desarrollar un modelo de análisis detallado del software, que incluya diagramas y documentación que respalden la comprensión y estructura del proyecto mediante el uso de UML. Además, se busca que el usuario consuma servicios REST utilizando APIs públicas, procesando datos en formatos como JSON o XML, e integrando funcionalidades adicionales en su software. Esto asegurará la correcta conexión y funcionalidad con los endpoints implementados en prácticas anteriores.

2.1 Objetivos Particulares

- Desarrollo del Modelo de Análisis: Se busca elaborar un modelo de análisis que describa en detalle las funciones, interacciones y características clave del software. Esto incluye la utilización de diagramas UML que faciliten la representación visual del sistema, permitiendo una mejor comprensión de su estructura y funcionamiento.
- Documentación UML: El objetivo es crear una documentación exhaustiva mediante diagramas UML, que incluya:
- Un Diagrama Entidad-Relación (ER) que represente las entidades y sus relaciones si se utiliza una base de datos.
- Un Diccionario de Datos que detalle los atributos y características clave de los datos manejados por el sistema.
- Un Diagrama de Casos de Uso que identifique los actores y sus interacciones con el sistema.
- Un Diagrama de Secuencias que ilustre el flujo del caso de uso más prioritario.
- Un Diagrama de Clases Conceptuales que defina las clases y sus relaciones en el dominio del software.
- Implementación de Funcionalidades Adicionales: Este objetivo se enfoca en permitir a los usuarios subir y modificar su foto de perfil, así como habilitar a

los administradores para gestionar las fotos de todos los usuarios. Además, se busca conectar el software con un servicio REST previamente creado, manteniendo la funcionalidad CRUD y gestionando roles y permisos.

- Consumo de APIs Públicas: Se pretende integrar al menos una API pública en el software, procesando y presentando la información al usuario. Esto incluye:
- La implementación de la Open Library API para permitir búsquedas sobre libros y autores.
- La integración de la TVMaze API para facilitar búsquedas sobre series y películas.
- Mejoras en la Usabilidad: Se busca implementar un sistema de registro y login mediante Google, mejorando así la experiencia del usuario al interactuar con la aplicación. Esta funcionalidad es obligatoria, mientras que la integración con otros servicios como Facebook es opcional.
- Documentación Final: El objetivo final incluye la preparación de un informe detallado que incluya todos los apartados requeridos, asegurando seguir las especificaciones formales indicadas para su presentación. Esto implica subir todo el código a un repositorio de GitHub y proporcionar capturas que muestren el sistema en funcionamiento, así como reflexiones sobre los retos enfrentados y los logros alcanzados durante el desarrollo del proyecto.

3 Tecnologías usadas

Para el desarrollo del proyecto de software, se utilizaron diversas tecnologías que permiten la creación de aplicaciones robustas y eficientes. A continuación se detallan las tecnologías y herramientas empleadas:

- Spring Boot con Maven
- Crear el proyecto Spring Boot
    - **Spring Initializr:** Se utilizó esta herramienta para generar un nuevo proyecto Spring Boot configurado con las siguientes especificaciones:
        - **Project:** Maven Project
        - **Language:** Java
        - **Spring Boot Version:** 3.3.4
        - **Group:** com.apiintegration
        - **Artifact:** api-consumer
        - **Packaging:** Jar
        - **Java Version:** 21
- Añadir dependencias necesarias

Se añadieron las siguientes dependencias principales al proyecto para habilitar diversas funcionalidades:

- **Spring Web:** Para crear aplicaciones web y RESTful.
- **Spring Security:** Para implementar la seguridad en la aplicación, gestionando la autenticación y autorización de usuarios.
- **Spring Data JPA:** Para facilitar la interacción con bases de datos mediante la implementación de la API JPA (Java Persistence API).

- **MySQL Driver:** Para permitir la conexión a una base de datos MySQL.
- **Thymeleaf:** Un motor de plantillas para generar vistas dinámicas en aplicaciones web.

- Otras Tecnologías y Herramientas

Además de Spring Boot, se utilizaron otras tecnologías y herramientas que complementan el desarrollo del proyecto:

- **Base de Datos MySQL:** Para almacenar datos de usuarios, libros, series y películas.
- **Postman o Retrofit:** Herramientas para realizar pruebas de los endpoints de las APIs y verificar su funcionamiento.
- **HTML/CSS/JavaScript:** Para el desarrollo del frontend, asegurando una interfaz amigable y responsiva.
- **GitHub:** Para el control de versiones y almacenamiento del código fuente del proyecto.

- APIs Públicas Integradas

Se integraron las siguientes APIs públicas para enriquecer las funcionalidades del software:

- **Open Library API:** Permite realizar búsquedas sobre libros y autores, facilitando el acceso a información bibliográfica.
- **TVMaze API:** Ofrece datos sobre series y películas, permitiendo a los usuarios explorar contenido audiovisual.

Estas tecnologías en conjunto permiten crear un sistema que no solo es funcional sino también escalable y seguro, facilitando la interacción del usuario con los servicios ofrecidos por las APIs integradas.

4 Desarrollo

4.1 Modelo de Análisis

4.1.1 Diagramas UML

4.1.1.1 Diagrama Entidad-Relación (ER)

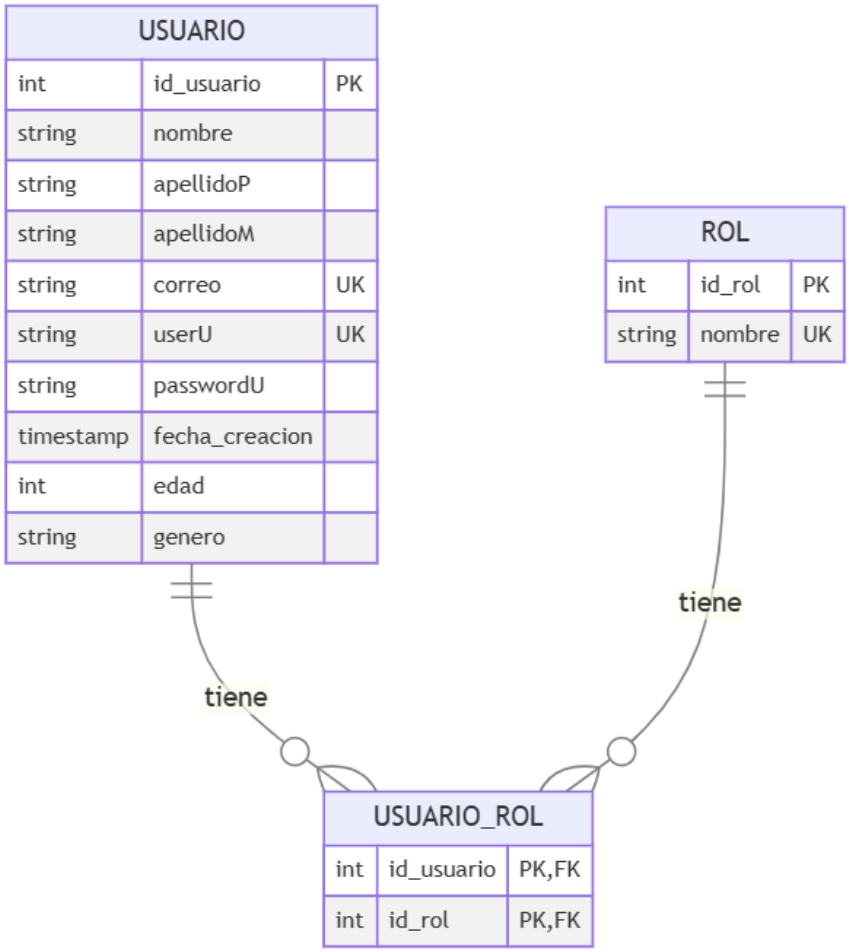

Figura 1 Diagrama Entidad-Relación

La entidad USUARIO con todos sus atributos:

- id_usuario como clave primaria (PK)
- campos únicos (UK): correo y userU
- resto de atributos: nombre, apellidos, password, fecha_creacion, edad y género

La entidad ROL con sus atributos:

- id_rol como clave primaria (PK)
- nombre como campo único (UK)

La tabla intermedia USUARIO_ROL que representa la relación muchos a muchos entre USUARIO y ROL:

- Compuesta por id_usuario e id_rol como claves primarias y foráneas
- Conecta las dos entidades principales

Las relaciones están representadas con la notación:

- ||--o{ que indica una relación uno a muchos
- La línea conecta USUARIO y ROL con USUARIO_ROL

4.1.1.2 Diccionario de Datos

Tabla: usuarioTabla: rol

Campo	Tipo	Longitud	Nulo	Restricción	Descripción
id_usuario	INT	11	No	PK, AUTO_INCREMENT	Identificador único del usuario
nombre	VARCHAR	64	No		Nombre del usuario
apellidoP	VARCHAR	64	No		Apellido paterno del usuario
apellidoM	VARCHAR	64	No		Apellido materno del usuario
correo	VARCHAR	64	No	UNIQUE	Correo electrónico del usuario
userU	VARCHAR	64	No	UNIQUE	Nombre de usuario para inicio de sesión
passwordU	VARCHAR	128	No		Contraseña del usuario
fecha_creacion	TIMESTAMP	-	No	DEFAULT CURRENT_TIMESTAMP	Fecha y hora de creación del registro
edad	INT	-	No		Edad del usuario
genero	VARCHAR	64	No		Género del usuario

Campo	Tipo	Longitud	Nulo	Restricción	Descripción
id_rol	INT	11	No	PK, AUTO_INCREMENT	Identificador único del rol
nombre	VARCHAR	64	No	UNIQUE	Nombre del rol (ej: ROLE_ADMIN)

Tabla: usuario_rol

Campo	Tipo	Longitud	Nulo	Restricción	Descripción
id_usuario	INT	11	No	PK, FK	ID del usuario, referencia a usuario(id_usuario)

id_rol	INT	11	No	PK, FK	ID del rol, referencia a rol(id_rol)

4.1.1.3 Diagrama de Casos de Uso

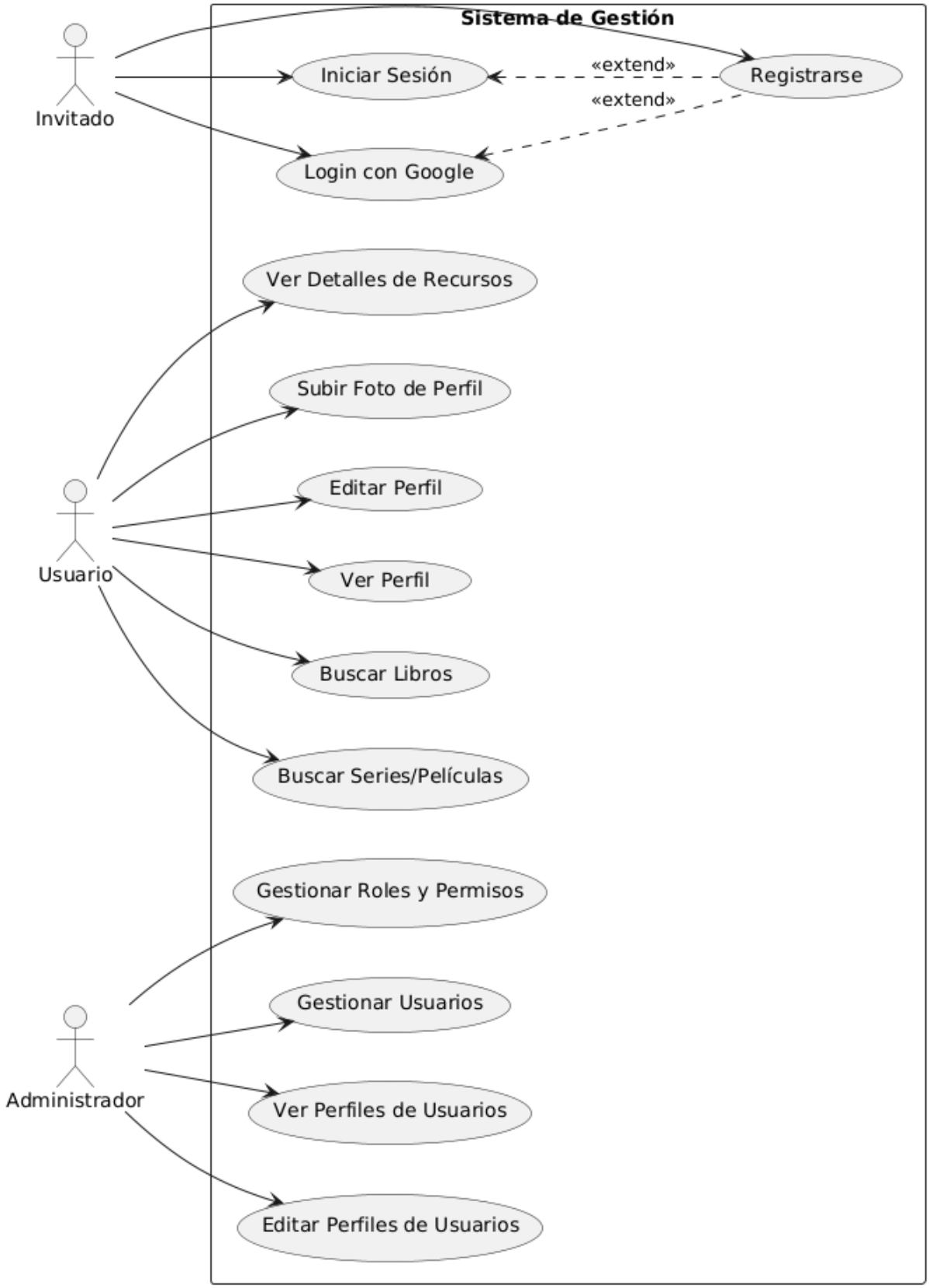

Figura 2 Diagrama del login.

Descripción del Diagrama de Casos de Uso

Actores del Sistema

1. **Invitado**
    - Representa usuarios que aún no han iniciado sesión
    - Capacidades limitadas de interacción con el sistema
2. **Usuario**
    - Tiene acceso a funcionalidades completas del sistema
    - Puede interactuar con la mayoría de los servicios
3. **Administrador**
    - Cuenta con privilegios especiales de gestión
    - Tiene control total sobre la plataforma

Casos de Uso por Categoría

Autenticación

- Iniciar Sesión
- Registrarse
- Registro/Login con Google

Perfil de Usuario

- Subir Foto de Perfil
- Editar Perfil
- Ver Perfil

Consumo de APIs

- Buscar Libros
- Buscar Series/Películas
- Ver Detalles de Recursos

Administración

- Gestionar Usuarios
- Ver Perfiles de Usuarios
- Editar Perfiles de Usuarios
- Gestionar Roles y Permisos

Relaciones entre Actores y Casos de Uso

Invitado

- Puede acceder a:
    - Iniciar Sesión
    - Registrarse
    - Registro/Login con Google

Usuario

- Puede acceder a:
    - Subir Foto de Perfil
    - Editar Perfil
    - Ver Perfil
    - Buscar Libros

- Buscar Series/Películas
- Ver Detalles de Recursos

Administrador

- Puede acceder a:
  - Gestionar Usuarios
  - Ver Perfiles de Usuarios
  - Editar Perfiles de Usuarios
  - Gestionar Roles y Permisos

4.1.1.4 Diagrama de Secuencias del Sistema

Diagrama de Secuencia: Registro de Usuario

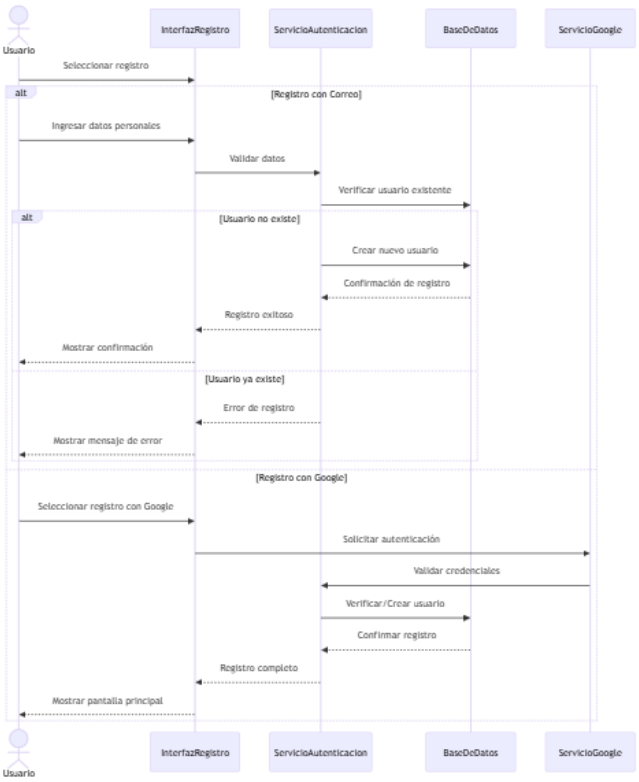

Figura 3 Diagrama de Secuencia: Registro de Usuario.

Descripción del Diagrama de Secuencia - Inicio de Sesión

Diagrama de Secuencia: Búsqueda y Recomendación de Libros

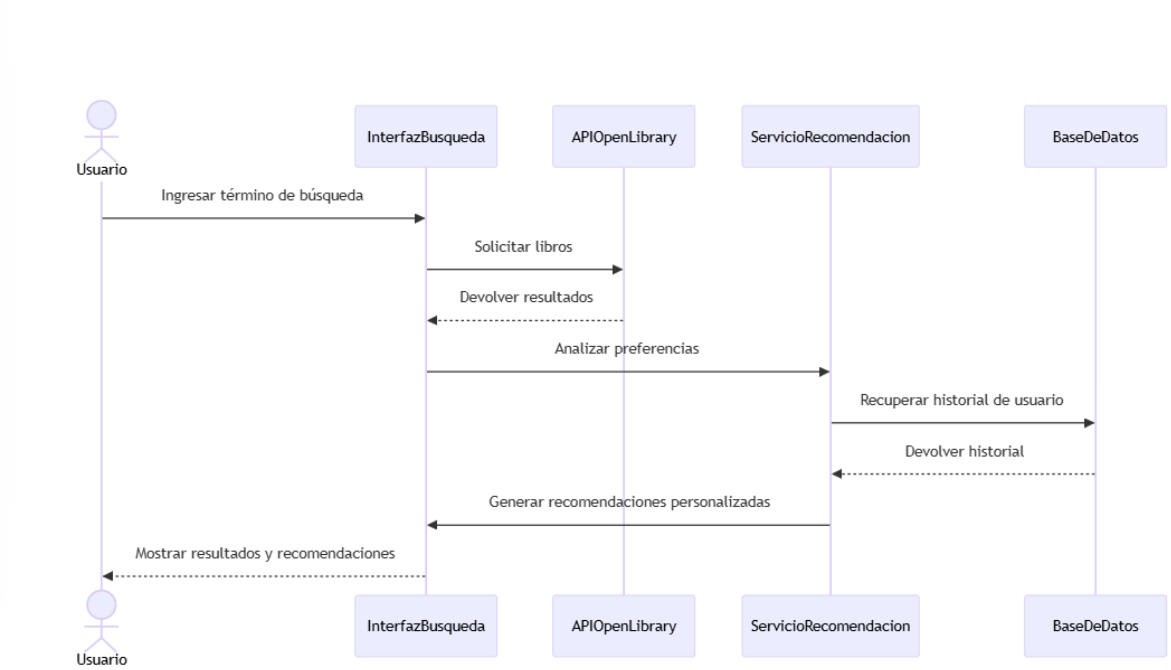

Figura 1 Búsqueda y Recomendación de Libros

Diagrama de Secuencia: Gestión de Usuarios por Administrador

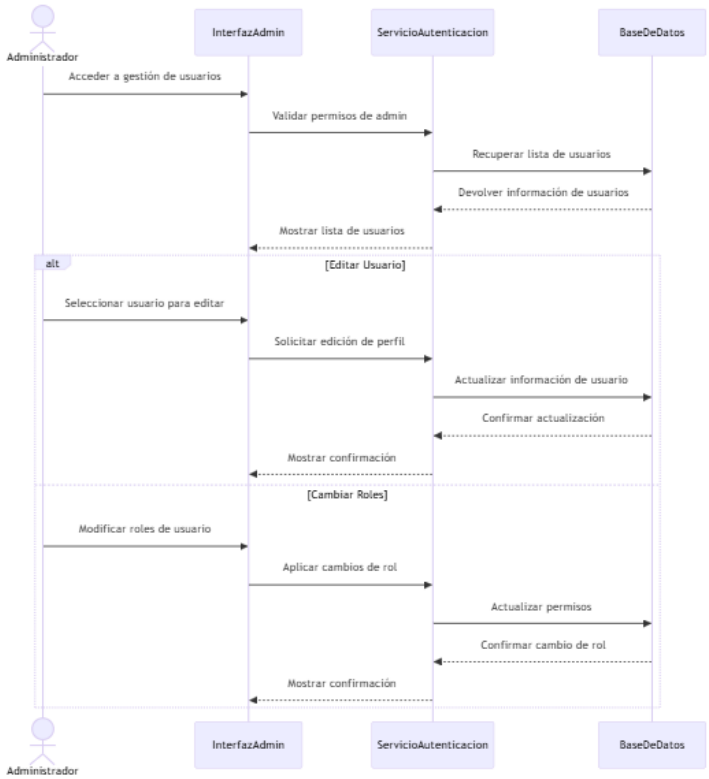

Figura 2 Diagrama de Secuencia: Gestión de Usuarios por Administrador

Diagrama de Secuencia: Búsqueda de Series y Películas

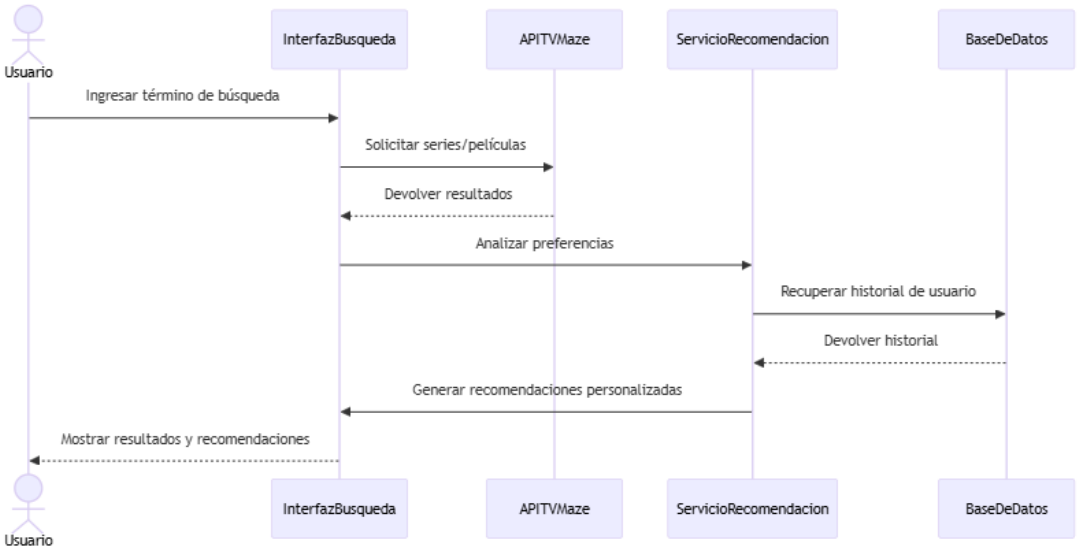

Figura 3 Diagrama de Secuencia: Búsqueda de Series y Películas.

Actores y Componentes

- Usuario: Actor que inicia el proceso de login
- Interfaz de Usuario (IU): Capa de presentación del sistema
- Controlador (C): Capa de lógica de negocio
- Modelo/BD (M): Capa de acceso a datos

Flujo de Secuencia Principal

Fase de Inicio

1. Usuario → IU: El usuario accede a la página de login del sistema

2. IU → Usuario: El sistema muestra el formulario de inicio de sesión

Fase de Entrada de Datos

3. Usuario → IU: El usuario ingresa sus credenciales:

- Nombre de usuario (userU)
- Contraseña (passwordU)

4. IU → Controlador: La interfaz envía los datos ingresados al controlador

Fase de Verificación

5. Controlador → Modelo: El controlador solicita los datos del usuario

6. Modelo → Controlador: La base de datos retorna la información del usuario

7. Controlador: Verifica las credenciales proporcionadas

Fase de Autorización

8. Controlador → Modelo: Solicita los roles asociados al usuario

9. Modelo → Controlador: Retorna los roles del usuario

Fase de Finalización

10. Controlador: Genera una nueva sesión para el usuario

11. Controlador → IU: Envía la confirmación de login exitoso

12. IU → Usuario: Redirige al usuario a la página principal del sistema

Consideraciones

- El diagrama muestra solo el escenario principal (camino feliz)
- Se asume que las credenciales son correctas
- La verificación incluye la validación de la contraseña hasheada
- La sesión generada incluye los roles del usuario para control de acceso

4.1.1.5 Diagrama de Clases Conceptuales

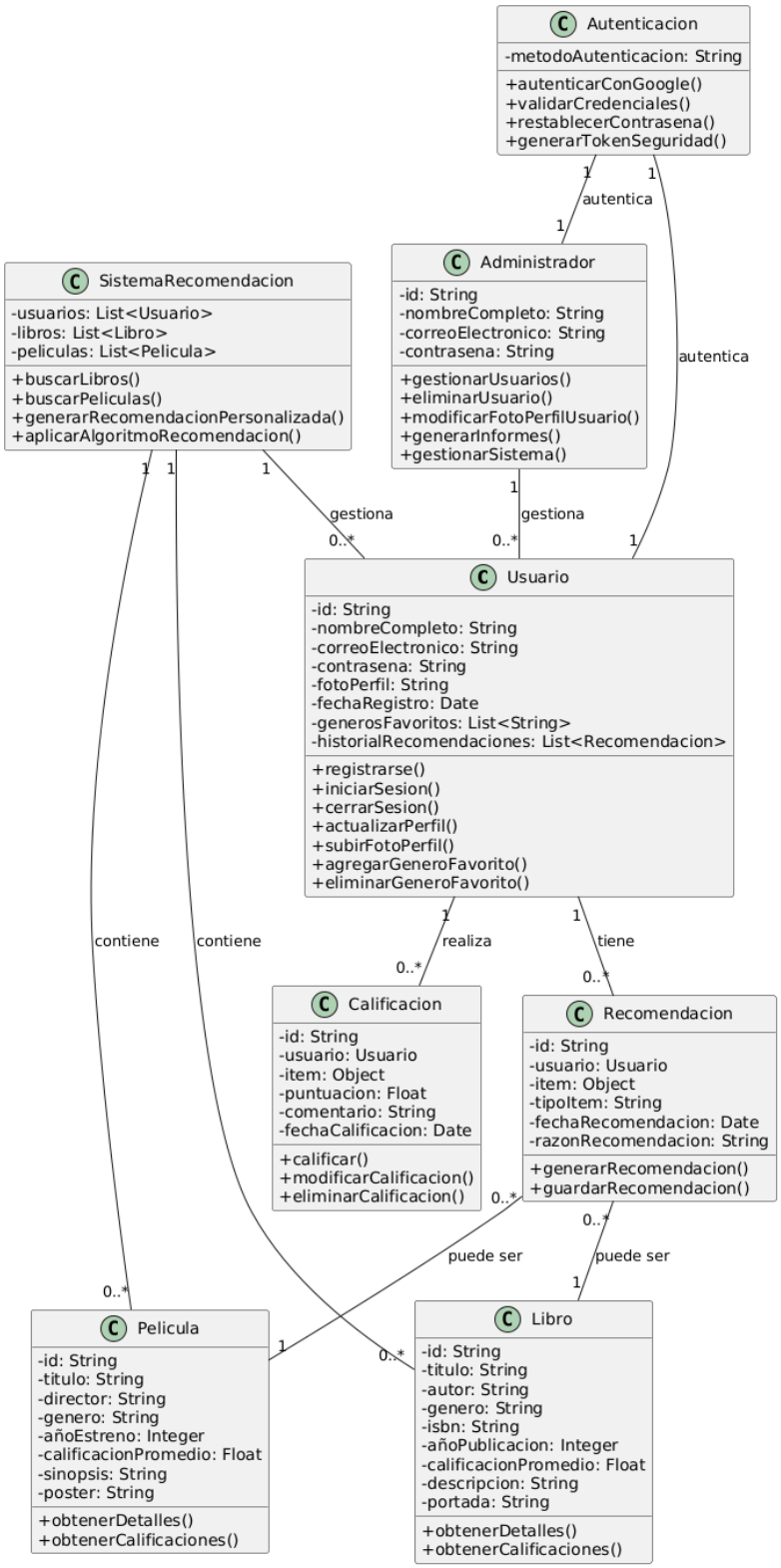

Figura 4 Diagrama de Clases Conceptuales

El diagrama incluye las siguientes clases principales:

1. **Usuario**: Representa los usuarios del sistema, con atributos como identificación, nombre, correo electrónico, contraseña, foto de perfil, fecha de registro, géneros favoritos e historial de recomendaciones. Incluye métodos para gestionar su perfil y cuenta.

2. **Administrador**: Una clase específica para usuarios con privilegios de administración, con capacidades para gestionar usuarios, sistemas y generar informes.

3. **Libro**: Contiene información detallada sobre libros, incluyendo título, autor, género, ISBN, año de publicación, calificación promedio, descripción y portada.

4. **Pelicula**: Similar a Libro, pero para películas, con atributos como título, director, género, año de estreno, calificación promedio, sinopsis y póster.

5. **Recomendación**: Gestiona las recomendaciones personalizadas, vinculando usuarios con libros o películas recomendadas.

6. **SistemaRecomendación**: Clase central que maneja la lógica de recomendación, conteniendo listas de usuarios, libros y películas.

7. **Calificación**: Permite a los usuarios calificar libros y películas, con métodos para crear, modificar y eliminar calificaciones.

8. **Autenticación**: Maneja los procesos de autenticación, incluyendo inicio de sesión con Google y gestión de credenciales.

El diagrama también muestra las relaciones entre estas clases, demonstrando cómo interactúan en el sistema.

Algunas características destacadas:

- Soporte para recomendaciones personalizadas
- Gestión de perfiles de usuario
- Funcionalidades de administración
- Autenticación con Google

- Capacidad de calificar y recomendar libros y películas

4.1.1.6 Modelo de Interfaz y Navegación

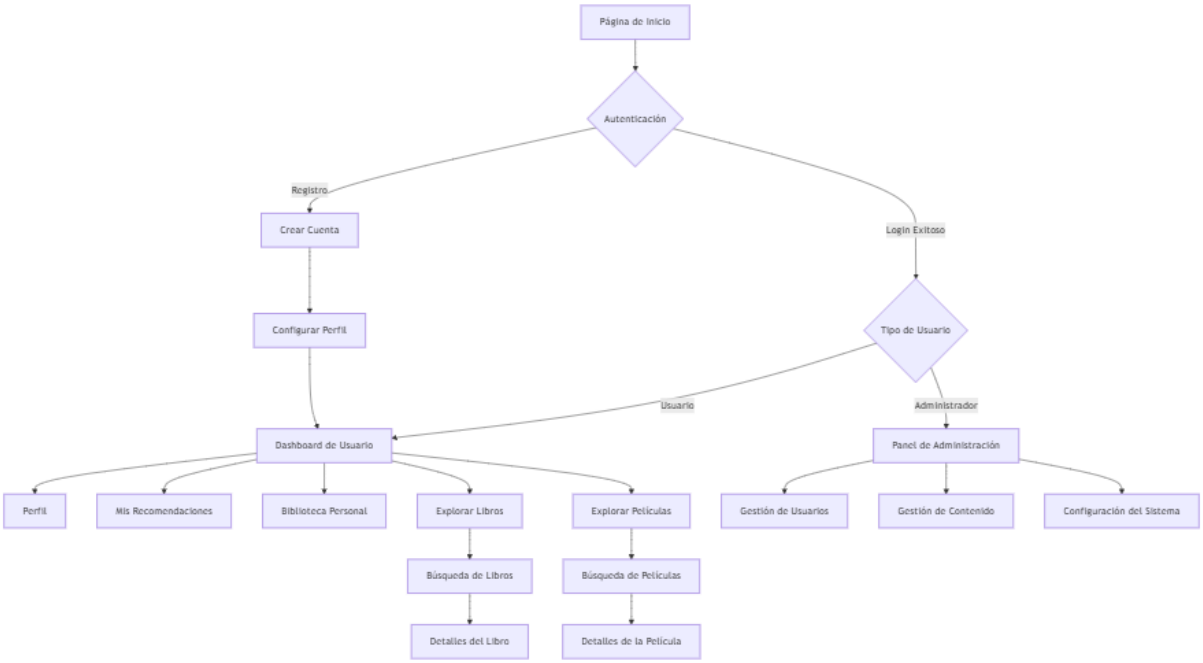

Figura 4 Modelo de Interfaz y Navegación.

Para el modelo de interfaz y navegación:

1. Un documento Markdown detallado que describe:
    - Flujos de navegación para usuarios y administradores
    - Pantallas principales
    - Flujos de interacción
    - Consideraciones de diseño
    - Tecnologías sugeridas
2. Un diagrama de navegación usando Mermaid que muestra visualmente:
    - Flujo de autenticación
    - Rutas de navegación para usuarios y administradores

- Principales secciones del sistema

El modelo está diseñado para un sistema de recomendación de libros y películas, con interfaces diferenciadas para usuarios regulares y administradores.

4.1.1.7 Diagramas de Robustez

Inicio de Sesión

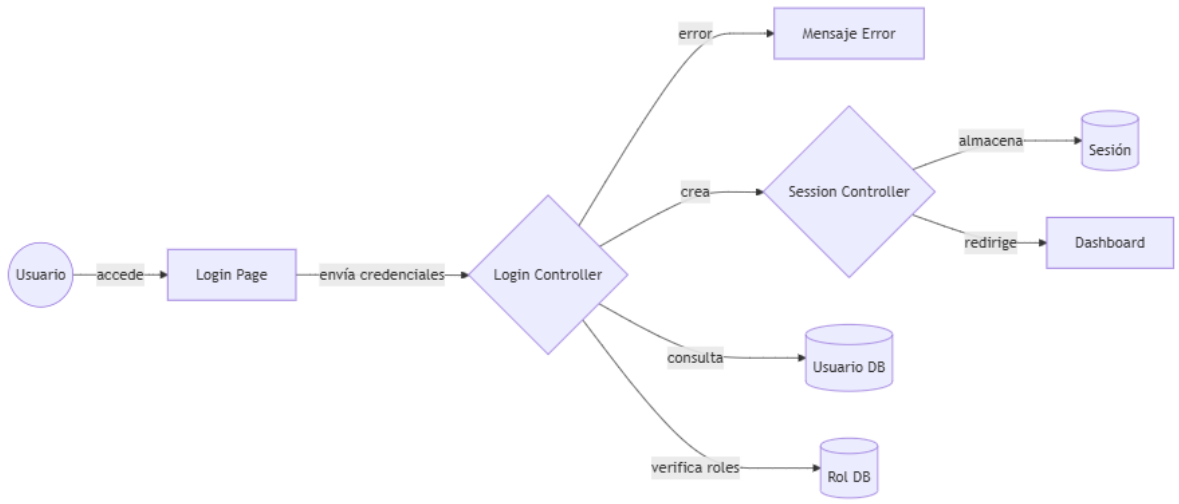

Figura 5 Diagrama de robustez del Inicio de Sesión.

Registro de Usuario

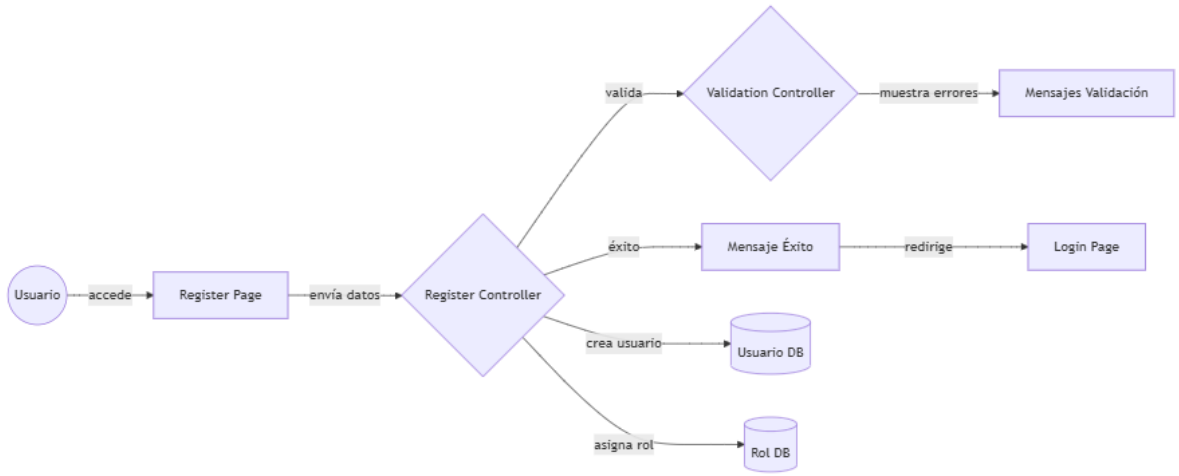

Figura 5 Diagrama de robustez del Registro de Usuario.

Gestión de Usuarios (Admin)

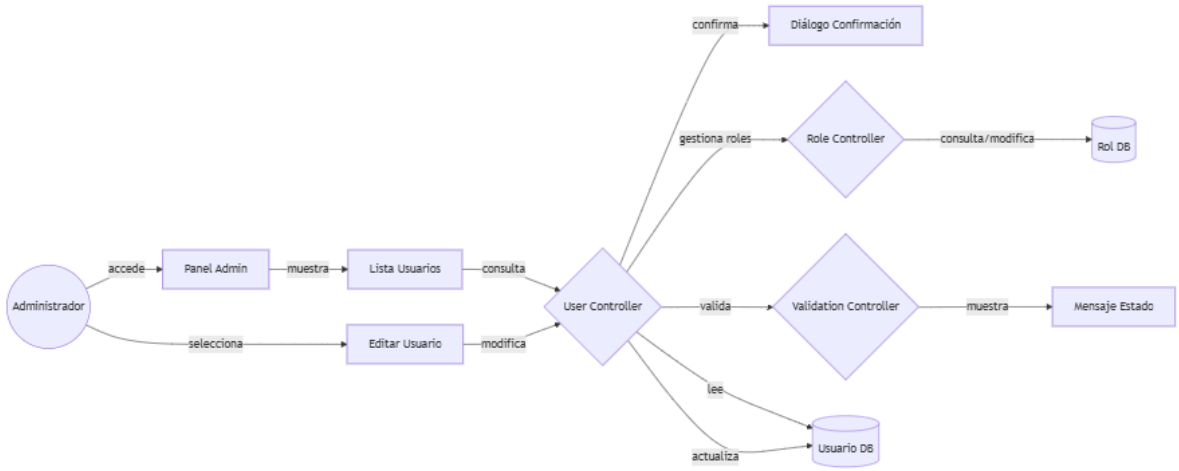

Figura 6 Diagrama de robustez del la Gestión de Usuarios (Admin) .

Diagrama de Robustez - Recomendación de Libros

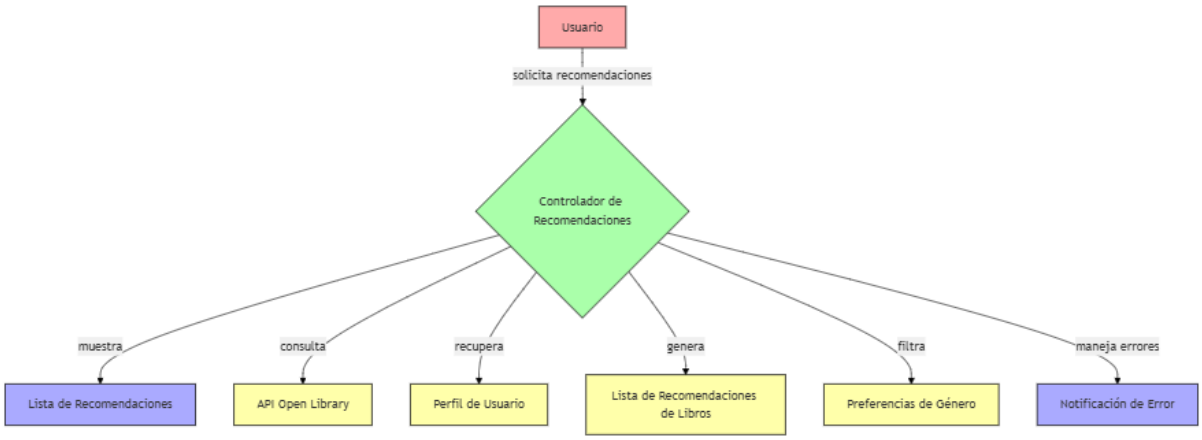

Figura 7 Recomendación de Libros

Diagrama de Robustez - Gestión de Usuarios por Administrador

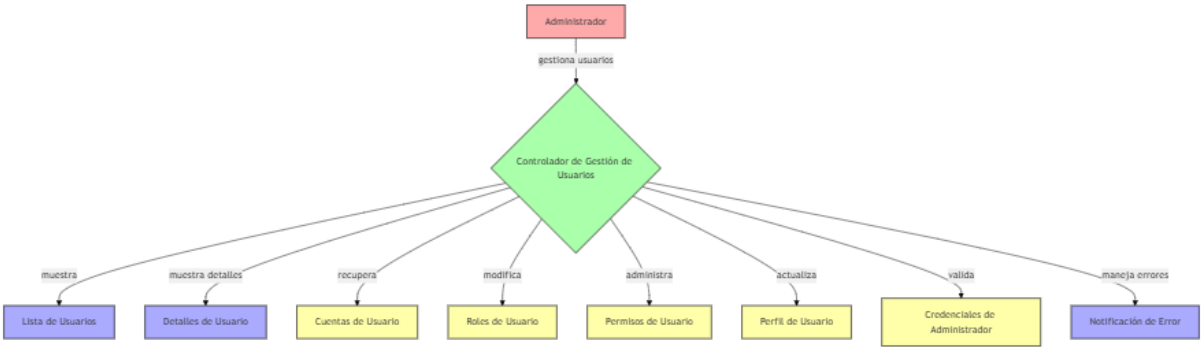

Figura 8 Gestión de Usuarios por Administrador

Diagrama de Robustez - Recomendación de Películas

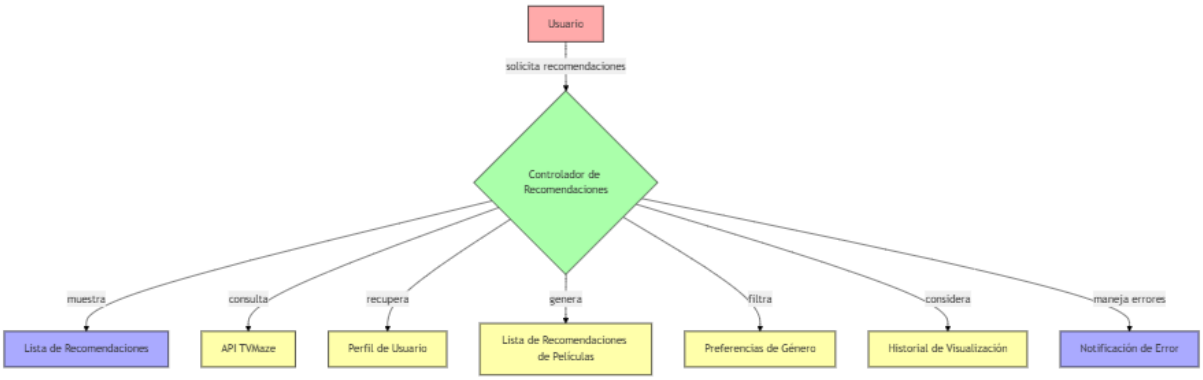

Figura 9 Diagrama de Robustez - Recomendación de Películas.

Los cuatro Diagramas de Robustez para el sistema de recomendación de libros y películas consisten en los siguiente:

1. Diagrama de registro de usuario
    - Muestra el proceso de registro de usuarios
    - Incluye validación, creación de cuentas y gestión de errores
2. Diagrama de recomendación de libros
    - Muestra cómo se generan las recomendaciones de libros
    - Integra la API de Open Library
    - Utiliza las preferencias del usuario y los datos del perfil
3. Diagrama de gestión de usuarios administradores
    - Muestra la capacidad del administrador para administrar cuentas de usuario
    - Muestra la lista de usuarios, los detalles, los roles y la administración de permisos
4. Diagrama de recomendación de películas
    - Similar a recomendación de libros
    - Utiliza la API de TVMaze
    - Tiene en cuenta el historial de visualización y las preferencias del usuario

Cada diagrama sigue el patrón de Boundary-Control-Entity (BCE):

- Actores (Rojo): Usuarios o Administradores
- Objetos de contorno (azul): interfaces de usuario e interacciones externas
- Procesos de control (verde): Controladores lógicos y de procesamiento

- Objetos de entidad (amarillo): datos y objetos persistentes

Los diagramas proporcionan una visualización clara de cómo interactúan los diferentes componentes del sistema durante los casos de uso clave. Muestran el flujo de información, las funciones de los diferentes componentes del sistema y cómo participan diversas entidades en el procesamiento de las solicitudes de los usuarios.

4.1.1.8 Diagrama de Clases de Diseño

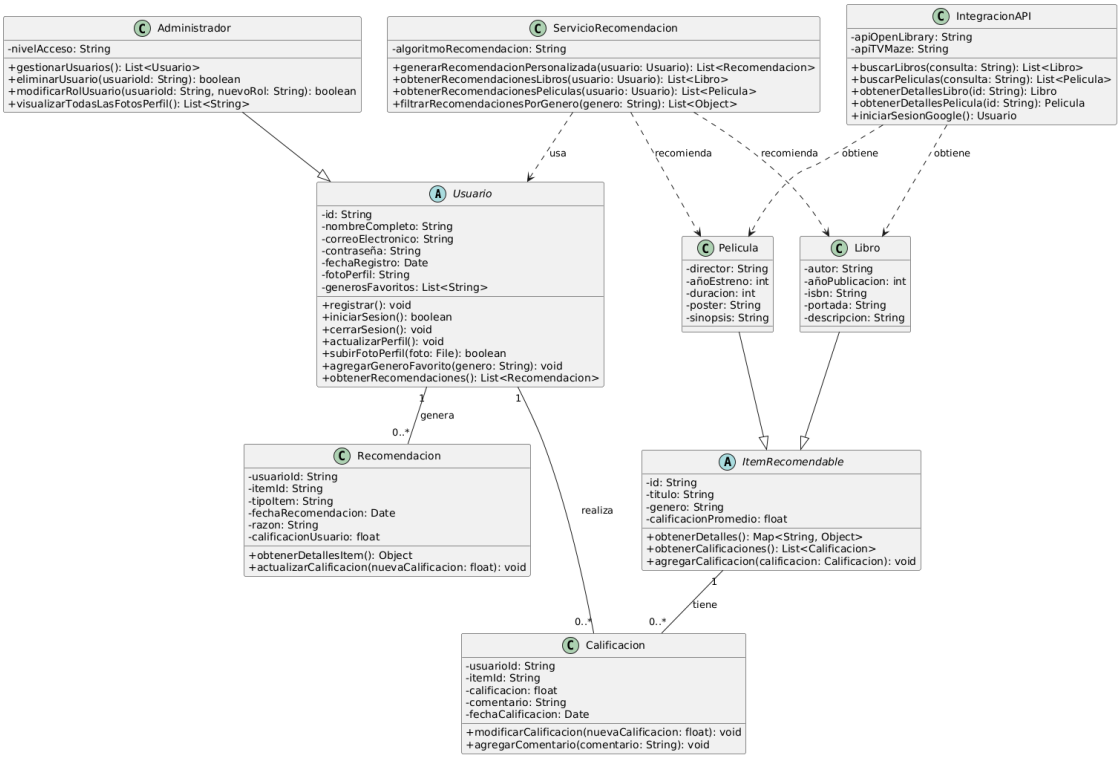

Figura 8 Diagrama de clases del Sistema de Recomendación de libros y películas.

El diagrama incluye las siguientes clases principales:

1. **Usuario**: Maneja la información y acciones básicas del usuario.
2. **Administrador**: Extiende la clase Usuario con funcionalidades adicionales de gestión.
3. **Libro**: Representa la información de un libro.
4. **Pelicula**: Representa la información de una película.
5. **Recomendacion**: Gestiona las recomendaciones personalizadas.

6. **Calificacion**: Maneja las calificaciones de libros y películas.
7. **ServicioRecomendacion**: Implementa la lógica de generación de recomendaciones.
8. **IntegracionAPI**: Gestiona la conexión con APIs externas como Open Library y TVMaze.

Algunas características destacadas:

1. **Clases Abstractas**:
    - Usuario: Clase base para usuarios
    - ItemRecomendable: Clase base para libros y películas
2. **Herencia**:
    - Administrador hereda de Usuario
    - Libro y Pelicula heredan de ItemRecomendable
3. **Relaciones**:
    - Asociaciones entre Usuario, Recomendacion, y Calificacion
    - Dependencias del ServicioRecomendacion con Usuario, Libro, y Pelicula
    - Dependencias de IntegracionAPI con Libro y Pelicula
4. **Multiplicidad**:
    - Un usuario puede tener múltiples recomendaciones y calificaciones
    - Un ítem recomendable puede tener múltiples calificaciones
5. **Métodos Clave**:
    - Gestión de usuarios
    - Generación de recomendaciones
    - Integración con APIs
    - Manejo de calificaciones

4.1.1.9 Diagrama de Interacción de Clases

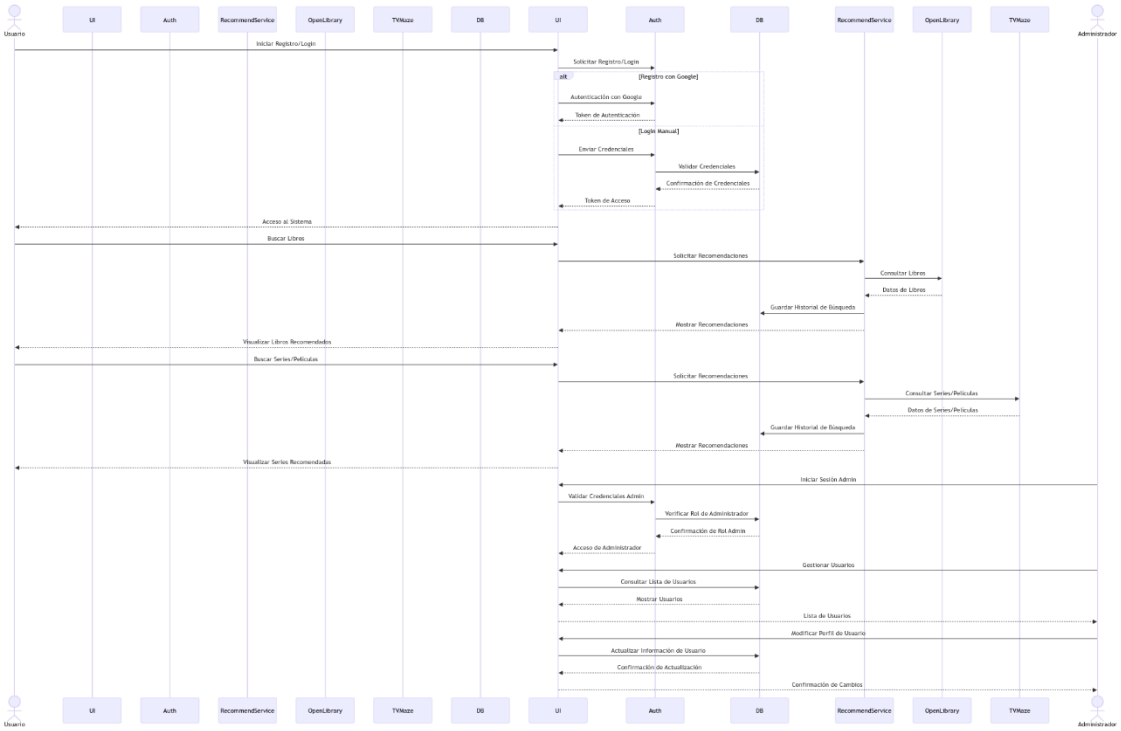

Figura 9 Diagrama de interacción.

El diagrama de interacción de clases incluye:

1. Flujo de Registro y Login
   - Registro con Google
   - Login manual
   - Validación de credenciales
2. Flujo de Recomendaciones de Libros
   - Búsqueda de libros
   - Consulta a Open Library API
   - Guardado de historial de búsqueda
3. Flujo de Recomendaciones de Series/Películas
   - Búsqueda de series/películas
   - Consulta a TVMaze API
   - Guardado de historial de búsqueda
4. Flujo de Administrador

- Login de administrador
- Validación de rol de administrador
- Gestión de usuarios
- Modificación de perfiles de usuario

Las principales clases/componentes que interactúan son:

- Usuario
- Interfaz de Usuario
- Servicio de Autenticación
- Servicio de Recomendaciones
- Open Library API
- TVMaze API
- Base de Datos

4.1.2 Documentación FURPS+

La documentación FURPS+ es un marco que ayuda a clasificar y describir los requisitos de calidad del software. A continuación, se detallan los aspectos relevantes para el proyecto de análisis y consumo de APIs.

4.1.2.1 Funcionalidad (F)

El software debe proporcionar las siguientes capacidades esenciales:

- **Manejo de usuarios:** Permitir el registro, inicio de sesión y gestión de perfiles de usuario.
- **Autenticación:** Implementar un sistema seguro de autenticación que incluya opciones como el registro y login mediante Google.

Consumo de APIs: Integrar y consumir datos de APIs públicas, como Open Library y TVMaze, permitiendo búsquedas de libros, autores, series y películas.

4.1.2.2 Usabilidad (U)

La usabilidad del software se enfoca en la experiencia del usuario:

- **Interfaz amigable:** La aplicación debe tener una interfaz intuitiva que facilite la navegación.
- **Facilidad de uso:** Los usuarios deben poder realizar acciones como registrarse, iniciar sesión y buscar información sin complicaciones.
- **Documentación clara:** Proveer guías o tutoriales que ayuden a los usuarios a entender cómo utilizar el sistema.

4.1.2.3 Confiabilidad (R)

Se deben implementar mecanismos para asegurar la confiabilidad del software:

- **Mecanismos de autenticación:** Usar protocolos seguros para la autenticación de usuarios, garantizando que solo los usuarios autorizados puedan acceder a sus cuentas.

- **Encriptación de contraseñas:** Implementar técnicas de encriptación para proteger las contraseñas almacenadas en la base de datos.

- **Manejo de errores:** Establecer un sistema robusto para manejar errores y excepciones, proporcionando mensajes claros al usuario en caso de fallos.

4.1.2.4 Rendimiento (P)

El rendimiento del software debe cumplir con ciertos estándares:

- **Tiempos de respuesta aceptables:** Definir tiempos máximos para las respuestas del sistema, especialmente al realizar búsquedas en APIs externas.

- **Uso eficiente de recursos:** Optimizar el uso de memoria y procesamiento para garantizar un funcionamiento fluido, incluso con múltiples usuarios concurrentes.

4.1.2.5 Soporte (S)

La mantenibilidad del sistema es crucial para su evolución:

- **Documentación técnica:** Proveer documentación detallada sobre el código y la arquitectura del sistema, facilitando futuras modificaciones.

- **Facilidad para realizar cambios:** Diseñar el software con patrones que permitan la adición o modificación de funcionalidades sin afectar otras partes del sistema.

4.1.2.6 Extensiones (+)

Consideraciones adicionales que pueden impactar el diseño e implementación del software:

- **Compatibilidad con plataformas específicas:** Asegurarse de que la aplicación funcione correctamente en diferentes navegadores y dispositivos móviles.

- **Uso de tecnologías particulares:** Considerar la integración con otras herramientas o servicios que puedan mejorar la funcionalidad del software, como servicios adicionales para análisis de datos o notificaciones.

Este marco FURPS+ proporciona una guía clara para evaluar y documentar los requisitos no funcionales del proyecto, asegurando que se cumplan las expectativas tanto del usuario como del desarrollador durante el proceso de desarrollo del software.

4.2 Implementación de Funcionalidades Adicionales

4.2.1 Foto de Perfil

4.2.1.1 Funcionalidades para Usuarios

La implementación de la funcionalidad de foto de perfil para usuarios se centra en el servicio FileUploadService.java, que maneja toda la lógica de carga y almacenamiento de imágenes. El ProfileController.java gestiona las operaciones relacionadas con el perfil del usuario, permitiendo la actualización y visualización de fotos. La interfaz de usuario se implementa a través de templates/user/profile.html, proporcionando una experiencia intuitiva para la gestión de fotos de perfil.

Principales componentes:

- Almacenamiento en /uploads/profiles/
- Manejo de carga de archivos en FileUploadService
- Interfaz de usuario en profile.html

```java

@Service
public class FileUploadService {
    private final String uploadDir = "uploads/profiles/";

    public String uploadProfilePhoto(MultipartFile file) throws IOException {
        // Create directory if it doesn't exist
        Files.createDirectories(Paths.get(uploadDir));

        // Generate unique filename
        String filename = UUID.randomUUID() + "_" + file.getOriginalFilename();
        Path filePath = Paths.get(uploadDir + filename);

        // Save file
        Files.copy(file.getInputStream(), filePath);

        // Return the URL path
        return "/uploads/profiles/" + filename;
    }
}
```

Figura 10 FileUploadService.java.

4.2.1.2 Funcionalidades para Administradores

La gestión administrativa de perfiles se implementa principalmente en AdminController.java, que proporciona funcionalidades avanzadas para la gestión de usuarios y sus perfiles. Los administradores tienen acceso a interfaces especializadas a través de templates/admin/users.html y edit-user.html, permitiendo una gestión completa de los perfiles de usuario, incluyendo la moderación de fotos y la edición de información.

```java
@GetMapping("/users")
public String listUsers(Model model) {
    List<User> users = userService.findAllUsers();
    model.addAttribute(attributeName:"users", users);
    return "admin/users";
}
```

Figura 11 Gestión administrativa de perfiles.

4.2.2 Conexión y Pruebas de API REST

4.2.2.1 Pruebas de Endpoints

4.2.2.2 Gestión de Roles y Permisos

La implementación de seguridad se basa en una estructura robusta definida en WebSecurityConfig.java, que configura todos los aspectos de seguridad y permisos. Los roles se definen en Role.java y ERole.java, mientras que UserDetailsServiceImpl.java maneja la autenticación y autorización de usuarios. RoleRepository.java gestiona la persistencia de roles en la base de datos.

```
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception
{
        http
            .csrf(csrf -> csrf.disable())
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/",
                                 "/auth/**",
                                 "/index",
                                 "/css/**",
                                 "/js/**",
                                 "/images/**",
                                 "/static/**",
                                 "/history/**",
                                 "/home",
                                 "/dashboard",
                                 "/index",
                                 "/movies",          // Permitir acceso a la
página de películas
                                 "/movies/search",   // Permitir búsquedas sin
autenticar
                                 "/movies/details/**",
                                 "/books",
                                 "/books/search",
                                 "/books/details/**").permitAll()
                .requestMatchers("/admin/**").hasAuthority("ROLE_ADMIN")
                .requestMatchers("/user/**").authenticated()
//                .requestMatchers("/home/**").authenticated()
                .requestMatchers("/history/**").authenticated()
                .requestMatchers("/search/**").authenticated()
                .requestMatchers("/recommendations/**").authenticated() //
Agregar esta línea
                .anyRequest().authenticated()
```

```
            .formLogin(form -> form
                .loginPage("/auth/login")
                .loginProcessingUrl("/auth/login")
                .defaultSuccessUrl("/home", true) // Cambia esto a /dashboard
                .permitAll()
            )
            .oauth2Login(oauth2 -> oauth2
            .loginPage("/auth/login")
            .defaultSuccessUrl("/home", true)
            .failureUrl("/auth/login?error")
            .userInfoEndpoint(userInfo -> userInfo
                .userService(oauth2Config.oauth2UserService()))
            )
            .logout(logout -> logout
                .logoutSuccessUrl("/auth/login?logout")
                .permitAll()
            );

        return http.build();
    }
```

Figura 12 WebSecurityConfig.java.

4.3 Consumo de APIs Públicas

4.3.1 Integración de APIs

4.3.1.1 Open Library API

La integración con Open Library se implementa principalmente a través de OpenLibraryService.java, que maneja todas las comunicaciones con la API externa. BookController.java gestiona las solicitudes de búsqueda y visualización de libros, mientras que las interfaces de usuario se implementan en los templates correspondientes.

Funcionalidades principales:

- Búsqueda integrada de libros y autores
- Visualización detallada de resultados
- Gestión de respuestas de la API

```java
public List<Show> getFeaturedShows() {
    try {
        // Obtener shows populares más actuales
        List<Long> featuredIds = Arrays.asList(...a:1L, 2L, 3L, 4L, 5L, 6L, 7L, 8L);
        List<Show> shows = new ArrayList<>();

        for (Long id : featuredIds) {
            try {
                String url = BASE_URL + "/shows/" + id;
                ResponseEntity<Show> response = restTemplate.getForEntity(url, respo
                if (response.getStatusCode().is2xxSuccessful() && response.getBody()
                    shows.add(response.getBody());
                }
                logger.info(format:"Fetching show from: {}", url); // Añadir log

                Show show = restTemplate.getForObject(url, responseType:Show.class);
                if (show != null) {
                    shows.add(show);
                }
            } catch (Exception e) {
                logger.error(format:"Error fetching show with ID {}: {}", id, e.getM
                continue; // Continuar con el siguiente show si hay error
            }
        }
        if (shows.isEmpty()) {
            logger.warn(msg:"No shows were retrieved from TVMaze API {}");
        }
        return shows.stream()
                    .filter(show -> show.getImage() != null && show.getImage().getMed
                    .collect(Collectors.toList());
    } catch (Exception e) {
        logger.error(msg:"Error fetching featured shows: ", e);
        return new ArrayList<>();
    }
}
```

```java
public Show getShowById(Long id) {
    try {
        String url = BASE_URL + "/shows/" + id;
        logger.info(format:"Fetching show from URL: {}", url);

        ResponseEntity<Show> response = restTemplate.getForEntity(url, responseType:S

        if (response.getStatusCode().is2xxSuccessful() && response.getBody() != null)
            logger.info(format:"Successfully retrieved show with ID: {}", id);
            return response.getBody();
        } else {
            logger.warn(format:"No show found with ID: {}", id);
            return null;
        }
    } catch (RestClientException e) {
        logger.error(format:"Error getting show by ID {}: {}", id, e.getMessage());
        throw new RuntimeException(message:"Error fetching show details", e);
    }
}
```

Figura 13 búsqueda y visualización de libros

4.3.1.2 TVMaze API

La implementación del consumo de la API de TVMaze se realiza a través de TVMazeService.java y sus controladores asociados. Los servicios proporcionan funcionalidades completas para la búsqueda y visualización de series y películas, con interfaces de usuario dedicadas en los templates correspondientes.

Componentes clave:

- Servicios especializados para series y películas
- Interfaces de búsqueda personalizadas
- Sistema de visualización de detalles

```java
public List<Book> getFeaturedBooks() {
    try {
        List<Book> books = searchBooks(query:"bestseller");
        return books.stream()
                    .limit(maxSize:4)
                    .collect(Collectors.toList());
    } catch (Exception e) {
        logger.error(msg:"Error fetching featured books: ", e);
        return new ArrayList<>();
    }
}
```

```java
public List<Book> searchBooks(String query) {
    try {
        String encodedQuery = URLEncoder.encode(query, StandardCharsets.UTF_8.toString
        String url = "https://openlibrary.org/search.json?q=" + encodedQuery;

        ResponseEntity<String> response = restTemplate.getForEntity(url, responseType:
        JsonNode root = new ObjectMapper().readTree(response.getBody());
        JsonNode docs = root.get(fieldName:"docs");

        List<Book> books = new ArrayList<>();
        if (docs.isArray()) {
            for (JsonNode doc : docs) {
                Book book = new Book();
                book.setTitle(doc.path(fieldName:"title").asText(defaultValue:"Unknown
                book.setAuthor(doc.has(fieldName:"author_name") ?
                    doc.get(fieldName:"author_name").get(index:0).asText() : "Unknown
                book.setIsbn(doc.has(fieldName:"isbn") ?
                    doc.get(fieldName:"isbn").get(index:0).asText() : "");

                if (doc.has(fieldName:"cover_i")) {
                    String coverId = doc.get(fieldName:"cover_i").asText();
                    book.setCoverUrl("https://covers.openlibrary.org/b/id/" + coverId
                }

                books.add(book);
            }
        }
        return books;
    } catch (Exception e) {
        logger.error(msg:"Error searching books: ", e);
        return new ArrayList<>();
    }
}
```

Figura 14 Búsqueda y visualización de series y películas.

4.3.2 Funcionalidades Adicionales

4.3.2.1 Registro y Login con Google

La implementación del consumo de la API de Google se realiza a través de OAuth2Service.java y sus controladores asociados. Los servicios proporcionan funcionalidades completas para la visualización de usuarios, con interfaces de usuario dedicadas en los templates correspondientes.

```
@Override
public OAuth2User loadUser(OAuth2UserRequest userRequest) throws OAuth2Authentication
    OAuth2User oauth2User = super.loadUser(userRequest);

    try {
        return processOAuth2User(userRequest, oauth2User);
    } catch (Exception ex) {
        logger.error(msg:"Error processing OAuth2 user", ex);
        throw new OAuth2AuthenticationException(ex.getMessage());
    }
}

private OAuth2User processOAuth2User(OAuth2UserRequest userRequest, OAuth2User oauth2
    // Extraer información del usuario de Google
    String email = oauth2User.getAttribute(name:"email");
    String name = oauth2User.getAttribute(name:"name");
    String pictureUrl = oauth2User.getAttribute(name:"picture");

    if (email == null) {
        throw new OAuth2AuthenticationException(errorCode:"Email not found from OAuth
    }

    User user;

    if (!userService.existsByEmail(email)) {
        user = createNewUser(email, name, pictureUrl);
    } else {
        user = userService.findByEmail(email)
                .orElseThrow(() -> new OAuth2AuthenticationException("User not found
        updateExistingUser(user, name, pictureUrl);
    }

    return oauth2User;
}
```

Figura 15 Registro y Login con Google.

4.4 Código Fuente

4.5 Capturas de Pantalla del Sistema en Funcionamiento

4.5.1.1 Foto de Perfil

Usuario

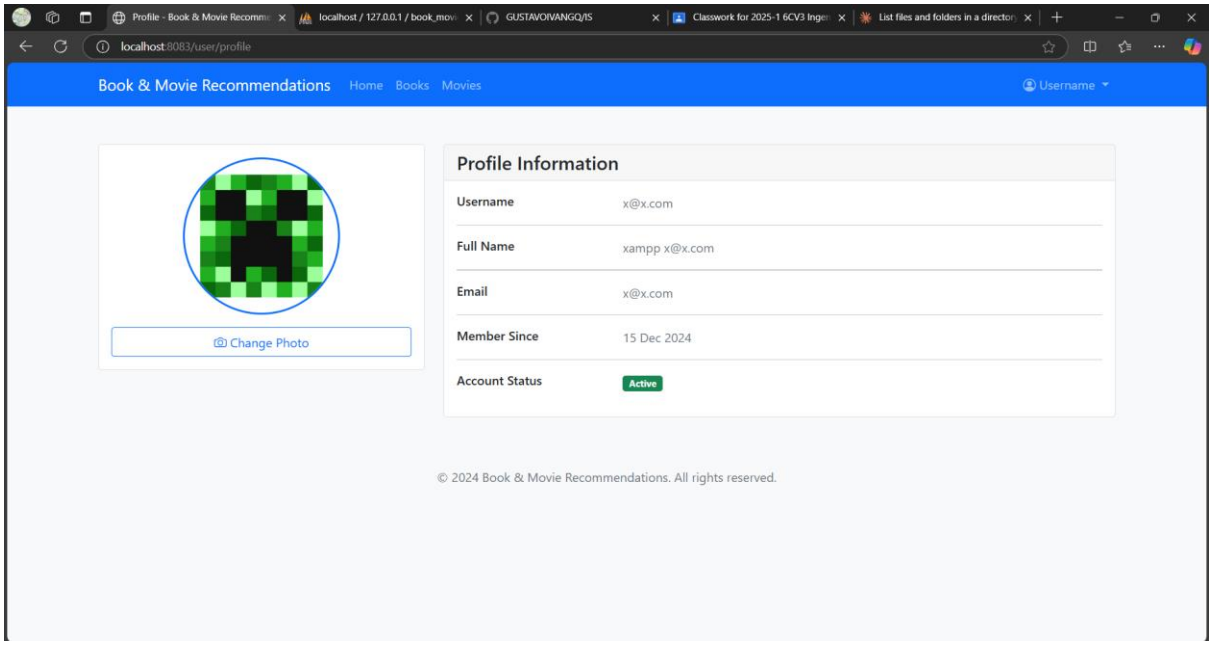

Figura 16 Foto de perfil de usuario.

Administrador

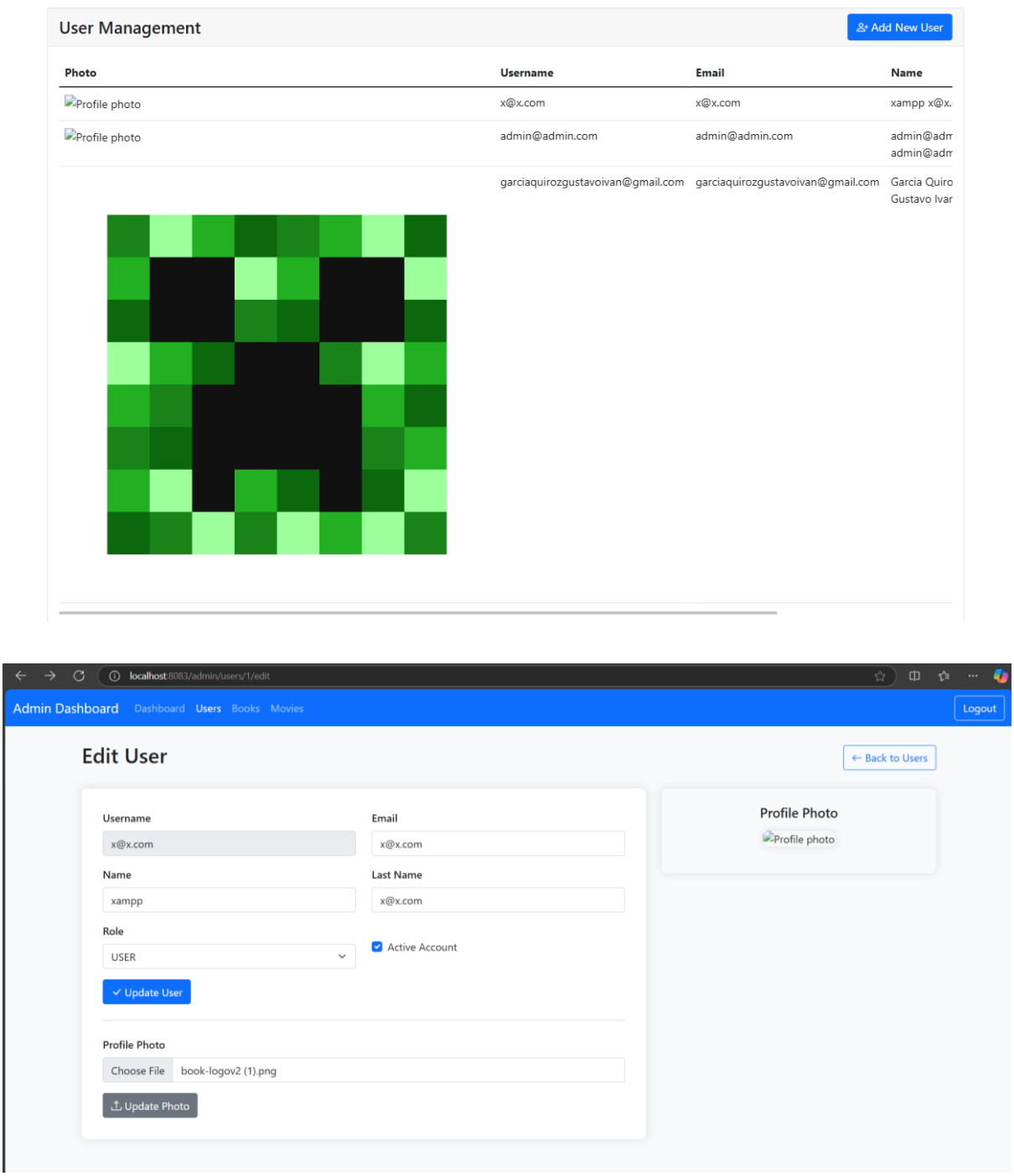

Figura 17 Vistas del administrador

Open Library

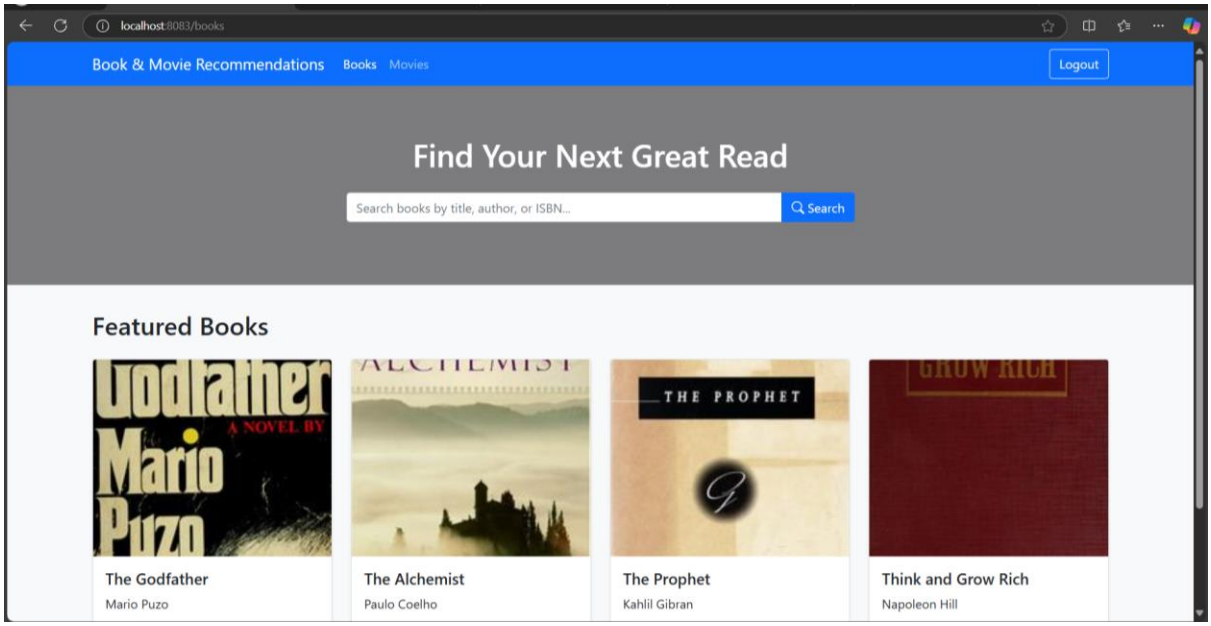

Figura 18 Vistas de libros de la Open Library API.

TVMaze

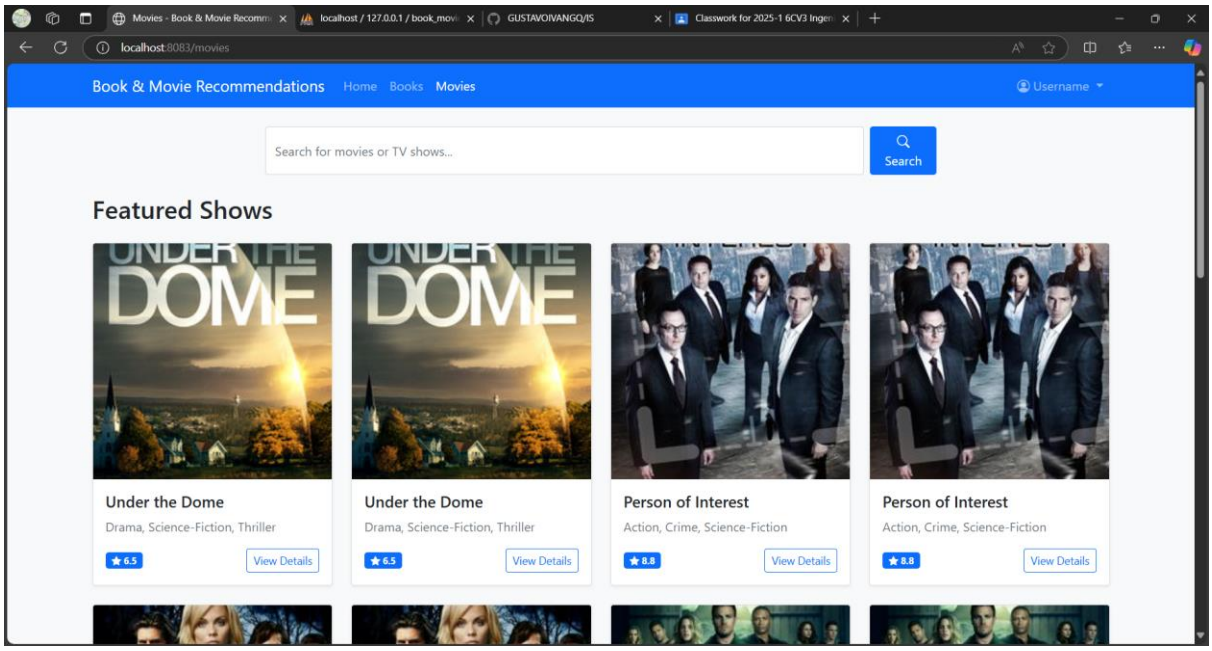

Figura 19 Vistas de series y películas de la TVMaze API.

Registro y login con Google

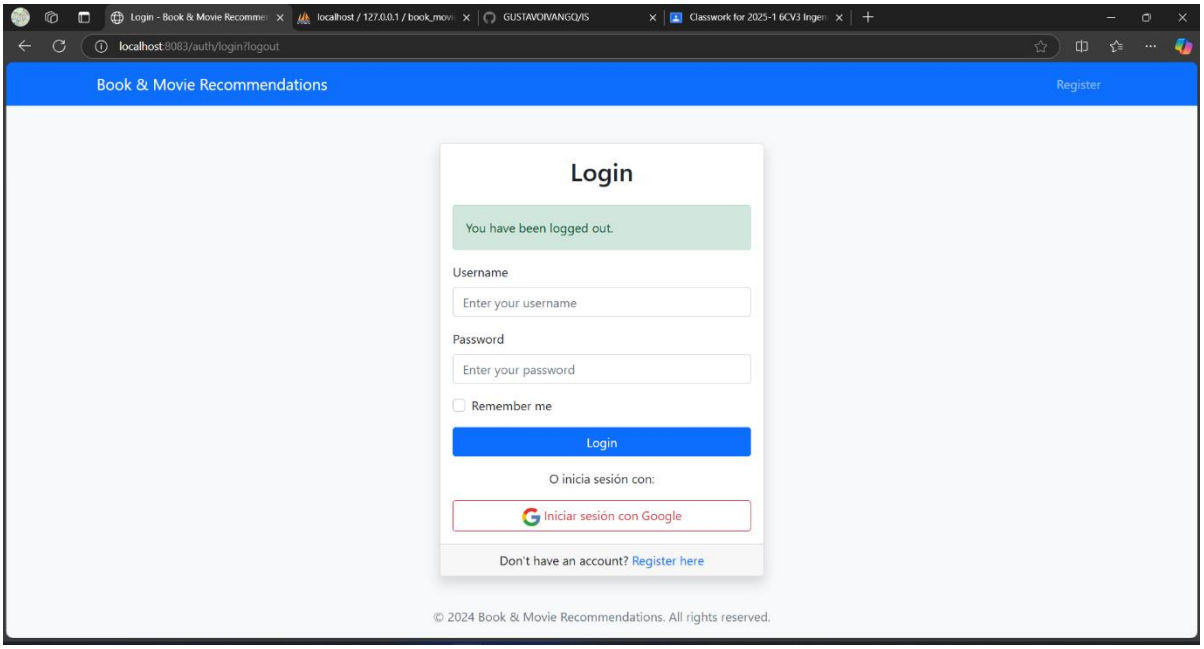

Figura 20 Registro y login con Google.

6 Conclusiones

La realización de la Práctica 3 de Modelo de Análisis y Consumo de APIs representó un desafío integral para comprender y aplicar conceptos avanzados de ingeniería de software. El desarrollo de los diversos diagramas UML nos permitió visualizar de manera estructurada y sistemática la arquitectura y funcionalidades del proyecto, revelando la importancia de una planificación detallada antes de la implementación de cualquier solución tecnológica.

La documentación FURPS+ resultó fundamental para analizar el software desde múltiples perspectivas, no solo considerando sus funcionalidades técnicas, sino también aspectos cruciales como usabilidad, confiabilidad, rendimiento y potencial de soporte. Este modelo de análisis nos obligó a pensar más allá del código, considerando la experiencia del usuario y la escalabilidad del sistema, elementos esenciales en el desarrollo de software profesional.

La integración de APIs públicas demostró la capacidad de extender las funcionalidades de un sistema mediante servicios externos, ampliando las posibilidades de nuestra aplicación. Trabajar con APIs como Open Library y TVMaze nos permitió comprender los desafíos de la integración de servicios REST, el manejo de datos JSON y la importancia de una comunicación eficiente entre diferentes plataformas tecnológicas.

La implementación de funcionalidades como la gestión de fotos de perfil y el registro con Google representó un avance significativo en la complejidad de nuestro proyecto. Estas características no solo mejoraron la experiencia del usuario, sino que también nos expusieron a técnicas de autenticación modernas y prácticas de desarrollo centradas en el usuario.

Los retos principales incluyeron la correcta interpretación de los requisitos, la sincronización entre diferentes componentes del sistema y la gestión precisa de roles y permisos. Cada obstáculo superado se convirtió en una oportunidad de aprendizaje, reforzando nuestra comprensión de los principios de la ingeniería de software y las mejores prácticas en el desarrollo de aplicaciones.

Finalmente, esta práctica reafirmó la importancia de una documentación clara, completa y bien estructurada. Los diagramas UML, la documentación FURPS+ y los diversos modelos de análisis no solo son requisitos académicos, sino herramientas fundamentales que facilitan la comunicación entre equipos de desarrollo, mejoran la comprensión del sistema y sientan las bases para una implementación exitosa.

8 Bibliografía APA

- Pressman RS. INGENIERIA DE SOFTWARE.; 2010.
- Sommerville I, Velázquez SF. Ingeniería de software.; 2011.
- Sommerville, I. (2015). Ingeniería de Software. Pearson Educación.
- Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language User Guide. Addison-Wesley.
- Microsoft Corporation. (2023). Microsoft Visio. Microsoft. Recuperado de https://www.microsoft.com
- JetBrains. (2023). IntelliJ IDEA. JetBrains. Recuperado de https://www.jetbrains.com/idea/
- StarUML. (2023). StarUML User Guide. MKLab. Recuperado de http://staruml.io/