Departamento de Computación
Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
MENCIÓN EN TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Análisis, Diseño e Implementación de una Aplicación Web Java MVC para la Puntuación y Recomendación de Películas

Estudiante: Roberto López Guntiñas
Director/a/es/as: José Losada Pérez

A Coruña, 5 de septiembre de 2019.

A mi madre

Agradecimientos

Me gustaría agradecer a toda mi familia el apoyo constante que he recivido para poder llegar hasta aquí. A mi padre en especial. También agradezco el impulso final necesario que me han dado mis amigos para poder terminar este trabajo. Además, doy las gracias al director de este trabajo, José Losada Pérez, por todos sus consejos, ideas y ayuda.

Gracias.

Resumen

El objetivo de este proyecto es el diseño e implementación de una aplicación web que permitirá a sus usuarios consultar información general sobre películas. Además, podrán puntuar y recomendar películas a otros usuarios.

La aplicación resultante será una red social basada en películas. Permitirá conocer las opiniones de otros sobre películas que ya ha visto, o permitirá a sus usuarios descubrir nuevas que ver.

Los usuarios administradores podrán gestionar las películas y otros usuarios registrados en la aplicación. Además, también se les permitirá registrar nuevos.

La metodología empleada para el desarrollo de este proyecto ha sido una versión simplificada del *Proceso Unificado de Desarrollo Software*. Se trata de un desarrollo iterativo e incremental en el que, en cada iteración se realiza un ciclo completo de las fases de análisis, diseño, implementación y pruebas para ir añadiendo nuevas funcionalidades progresivamente hasta alcanzar el objetivo establecido.

Para la fase de desarrollo del proyecto se van a utilizar tecnologías enfocadas a una arquitectura Modelo-Vista-Controlador (*MVC*). Una de ellas será la tecnología *Java EE* con apoyo en el *framework Spring*, concretamente en la herramienta *Spring Boot* para simplificar el desarrollo del subsistema que expone e implementa las operaciones del servicio. En la parte de la interfaz de usuario se va a utilizar la librería de *Javascript* de código abierto *React*

Abstract

The objective of this project is to design and implement a web application that allow users consult general information about movies. In addition, users will be able to rate and recommend movies to other users.

The resulting application will be a movie-based social network. It will allow users to know the opinions of other users about movies that they have already watched or it will also allow users to discover new movies to watch.

Administrator users can manage the movies and users registered in the application. In addition, they will also be allowed to register new ones.

The methodology used for the development of this project has been a simplified version of the *Unified Software Development Process*. It is an iterative and incremental development in which, in each iteration, a complete cycle of the phases of analysis, design, implementation and testing is carried out to gradually add new functionalities until reaching the established objective.

For the development phase of the project, technologies focused on an *Model-View-Controller* (MVC) architecture will be used. One of them will be the *Java EE* technology with help of the *Spring Framework*. More specifically, the *Spring Boot* tool will be used to ease the development of the server part. In the part of the user interface it will be used the open-source *JavaScript* library *React*.

Palabras clave:

- Película
- MVC
- Java
- Maven
- React
- API
- Web
- Spring

Keywords:

- Movie
- MVC
- Java
- Maven
- React
- API
- Web
- Spring

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1 Motivación

En la actualidad existe un creciente auge en el consumo de contenido audivisual en todo el mundo. Esto es debido, en parte, a la aparición y popularización de las principales plataformas de contenido multimedia en *streaming* y a su facilidad de uso. Esto sumado también a la popularización de las redes sociales, hace que muchos de los temas de debate en línea tenga que ver o estén relacionados con películas y series de actualidad.

Dentro de este marco, este proyecto pretende crear una aplicación web en la que se pueda consultar información de películas, recomendarlas a amigos o conocidos, puntuarlas, dar tú opinión y conocer la de otros.

Existen multitud de alternativas en la web que permiten la consulta de información de películas y todo el elenco que ha trabajado en ellas, como pueden ser: **IMDb**, quizás la más conocida, **Rotten Tomatoes** o la web en español más utilizada, **Filmaffinity**, etc... Pero lo que se busca con este proyecto es crear una aplicación web de películas más centrada en sus usuarios. Conocer qué películas ha visto un usuario, cuáles quiere ver, qué le ha parecido una película determinada, etc...

1.2 Alcance y objetivos

Este proyecto pretende crear la base de un sistema que consiste en una red social en la que sus usuarios puedan obtener información clara y relevante sobre películas. Además, también se pretende que los usuarios puedan registrar su opinión acerca de las películas que ha visto y consultar las opiniones de otros, para poder tener una perspectiva nueva o descubrir nuevas películas que ver.

A continuación se expondrán las funcionalidades más relevantes:

- **Gestión de usuarios**. Módulo que agrupa las operaciones relacionadas con el alta de usuarios en el sistema, edición y eliminación. Existirán tres tipos de usuario: administrador, espectador y crítico.
- **Gestión de películas**. Agrupa las operaciones sobre películas. Entre ellas se encuentran las operaciones propias de un administrador como agreagar, editar y eliminar información sobre ellas. Además, también se podrán realizar acciones de búsqueda.
- **Gestión de críticas**. Dentro de la aplicación se les permitirá a los usuarios exponer su opinión sobre una película mediante la publicación de críticas sobre ella.
- **Gestión de puntuaciones**. En la aplicación también se permitirá puntuar una película con un valor del uno al diez. Calculando la media de todas las puntuaciones de los usuarios, se obtendrá la puntuación de la película en el sistema.
- **Gestión de seguidores**. Los usuarios de la aplicación tendrán la posibilidad de seguir a otros usuarios con gustos afines en películas.
- **Gestión de las actividades de los usuarios**. Los usuarios del sistema podrán ver en la página principal las actividades más recientes de otros a los que sigue. Estas actividades serán de tres tipos: un usuario a empezado a seguir a otro, un usuario ha puntuado una película o un usuario ha recomendado una película.
- **Gestión de búsquedas**. Permite la ejecución de búsquedas sobre las películas registradas en el sistema aplicando diferentes filtros.

1.3 Estructura de la memoria

La presente memoria se ha estructurado en 9 capítulos, 3 apéndices y el apartado de bibliografía. A continuación se explica brevemente su contenido:

- **Capítulo 1:** Capítulo en el que se introduce el trabajo, su alcance y sus motivaciones.
- **Capítulo 2:** Capítulo en el que se exponen y explican las principales herramientas y tecnologías utilizadas durante el desarrollo.
- **Capítulo 3:** Capítulo en el que se analiza la viabilidad del proyecto.
- **Capítulo 4:** Capítulo en el que se explica la arquitectura global del siustema y la metodología utillizada para el desarrollo.

- **Capítulo 5:** Capítulo en el que se realiza la planificación del proyecto estimando el coste de las tareas y simulando un proyecto real.
- **Capítulo 6:** Capítulo en el que se exponen todos los requisitos que debe cumplir el sistema para ser aceptado.
- **Capítulo 7:** Capítulo en el que se expone el diseño que se ha realizado para la aplicación.
- **Capítulo 8:** Capítulo en el que se explica la implementación relizada utilizando diversas herramientas. También se explican los tipos de pruebas realizadas.
- **Capítulo 9:** Capítulo en el que se desarrollan las conclusiones adquiridas al finaliza el proyecto y posibles líneas futuras de trabajo.
- **Anexo A:** Glosario de acrónimos.
- **Anexo B:** Glosario de términos.
- **Anexo C:** API REST.
- **Bibliografía**

Capítulo 2

Herramientas y tecnologías utilizadas

En este capítulo de la memoria se explicarán, de forma resumida, las tecnologías y herramientas que se han utilizado para la realización de este proyecto.

2.1 Lenguajes de programación

2.1.1 Java y Java EE

El lenguaje de programación Java [1] es un lenguaje de propósito general, concurrente, basado en clases y orientado a objetos. El lenguaje está diseñado para que sea lo suficientemente simple para que el mayor número de desarrolladores lo puedan usar de forma fluida. Es un lenguaje pensado para entornos de producción y no para entornos de investigación, por este motivo, el diseño ha evitado incluir características nuevas y no probadas.

El lenguaje de programación Java es un lenguaje de alto nivel. Incluye gestión automática del almacenamiento, normalmente utilizando un recolector de basura, para evitar los problemas de seguridad de la desasignación explícita (como ocurre con los lenguajes C o C ++). El lenguaje tampoco incluyen ninguna estructura insegura, como pueden ser el acceso a arrays sin comprobar el índice, ya que este tipo de estructuras podrían provocar un comportamiento indeseado del porgrama.

Java Platform, Enterprise Edition (Java EE) [2], es un plataforma de desarrollo de aplicaciones *Java* para el ámbito empresarial, que permite un desarrollo más rápido y sencillo. El objetivo principal de esta herramienta es proveer a los desarrolladores con una pontente colección de *APIs* para acortar el tiempo de desarrollo, reducir la complejidad de la aplicación y

mejorar su rendimiento.

La plataforma *Java EE* es desarrollada por un grupo de expertos a través del programa denominado *Java Comnunity Process*, creando *Java Specification Requests*, para definir las distintas tecnologías *Java*. Esto ayuda a asegurar los estándares de estabilidad y de compatibilidad entre plataformas.

2.1.2 JavaScript

JavaScript [3] es un lenguaje de programación ligero, interpretado y dotado con capacidades para la programación orientada a objetos. El proposito general del lenguaje es permitir inlcuir contenido ejecutable en las páginas *web*. Esto quiere decir que las páginas *web* ya no tiene que ser solamente *HTML* estático, si no que también pueden incluir programas que interactúan con el usuario, controlan el navegador y crean contenido *HTML* de forma dinámica.

Cuándo un interprete de *JavaScript* está *incustrado* en un navegador *web*, se denomina *client-side JavaScript* o *JavaScript* del lado cliente. Esta variante de *JavaScrtipt* es de lejos la más utilizada, pero no la única. En general, *client-side JavaScript* combina las capacidades de ejecutar *scripts* del interprete, con el *document object model (DOM)* definido por el navegador *web*.

ECMA define la versión estándar del lenguaje *JavaScript* en su especificación ECMA-262, de la cual se disponen diferentes versiones que se han ido actualizando y revisando a lo largo del tiempo. Mientras que la especificación del *DOM* ha sido publica por la *World Wide Web Consortium (W3C)* que estandariza las características que un navegador debe soportar en relación a su *DOM*.

2.1.3 CSS

CSS (Cascading Style Sheets o Estilo en Cascada) [4] es un lenguaje que permite controlar el aspecto o presentación de documentos electrónicos definidos con HTML o XHTML. Permite la separación entre contenido y su presentación, lo que presenta numerosas ventajas, por ejemplo obliga a crear documentos HTML/XHTML bien definidos y con significado completo. También mejora la accesibilidad del documento, reduce la complejidad del mantenimiento y permite visualizar el mismo contenido de forma correcta en infinidad de dispositivos diferentes.

Una vez creado el contenido, con CSS se pasa a definir el aspecto de cada elemento como su color, tamaño, tipografía, separación vertical y horizontal, posición de cada elemento dentro

de la página, etc.

2.1.4 LaTex

LaTex [5] es un sistema de preparación de documentos para la composición tipográfica de alta calidad. Normalmente se usa para documentos técnicos o científicos de tamaño mediano o grande pero es posible utilizarlo para cualquier forma de publicación

La filosofía de Latex es animar a los autores a no preocuparse demasiado por la apriencia de sus documentos y más en su contenido.

2.2 Frameworks y librerías

2.2.1 Spring

Spring Framework [6] es una plataforma Java ”ligera” que permite el desarrollo de aplicaciones Java centrándose en la propia aplicación, delegando en *Spring* el manejo de la infraestructura. Permite la inversión de control (IoC), la inyección de dependencias, la gestión de forma declarativa de la transaccionalidad, el acceso remoto a la lógica a través de RMI (*Java Remote Method Invocation*) o servicios web además de una variedad de opciones para la persistencia de datos. Permite también la programación orientada a aspectos (AOP) y dispone de un *framework* completo para crear aplicaciones MVC.

Spring esta diseñado para no ser intrusivo, haciendo que la lógica del código de tu aplicación generalmente no dependa del *framework* en si. Además es modular, es decir, te permite utilizar solamente las partes que necesites, sin tener que incorporar también el resto de módulos.

Spring está compuesto por distintas características organizadas en unos 20 módulos. Todos sus módulos están agrupados en 6 grandes grupos: *Core Container*, *Data Access/Integration*, *Web*, *AOP (Aspect Oriented Programming)*, *Instrumentation* y Test.

Los módulos que se han utilizado para este proyecto han sido:

- ***Core and Beans***. Agrupados en el *Core Container*, proporcionan las partes fundamentales del *framework* incluidas la IoC (Inversión de Control) y la inyección de dependencias. Incluyen también el *BeansFactory*, el cual es una sofisticada implementación del patrón factoría, que consigue eliminar la necesidad de *singletons* programáticos y

permite desacoplar la configuración y especificación de dependencias del código con lógica de negocio.

- ***Web***. Se encuentra recogido en la parte Web de *Spring*. Este módulo proporciona integración básica de características web como puede ser la incialización del contenedor de la inversión de control usando *servlet listeners*, un contexto de aplicación orientado a web y otras características de soporte remoto.

Spring Boot

Spring [7] es un framework "ligero" en términos de código de los componentes, pero es, o más bien era, pesado en términos de configuración. Pero esto ha ido cambiando a lo largo de los años de vida de *Spring*. Inicialmente, *Spring* se configuraba mediante XML (mucho). *Spring 2.5* introdujo el escaneo automático de componentes basado en anotaciones, lo que eliminó la mayor parte de la configuración explícita por XML para los componentes propios de la aplicación. *Spring 3.0* introdujo configración basada en Java, como una opción segura y refactorizable frente al XML.

Pero, aún incluyendo todos estas actualizaciones, no había forma de escapar de la configuración, ya fuera XML o basada en Java. Por ejemplo, ciertas caracteristicas de *Spring* como el manejo de la transaccionalidad, *Spring MVC*, librerías de terceros, servlets y filtros requieren configuración explícita para activarlas.

Toda esta configuración constituye lo que se denomina "fricción de desarrollo". Es decir, todo el tiempo que se emplea en esta, es tiempo que no se ha dediado al desarrollo de la aplicación en si, de la lógica de negocio. Otra parte importante que constituye esta "fricción de desarrollo", es el manejo de las dependecias y su versiones para que todas funcionen correctamente unas con otras. *Spring Boot* ha cambiado todo esto.

Spring Boot proporciona numerosas ventajas en el desarrollo de una aplicación *Spring*, pero las principales son las siguientes:

- **Configuración automática**. *Spring Boot* proporciona la configuración común entre la mayoría de las aplicaciones *Spring*.
- **Dependencias iniciales**. Se le dice a *Spring Boot* el tipo de funcionalidad que necesitas, y automaticamente se asegura de que las librerías necesarias se añaden al proyecto.
- **CLI (Command-Line Interface)**. Se trata de una característica opcional. Permite la construcción de aplicaciones solamente con código de aplicación, no es necesario el

tradicional *project build*.

- ***The Actuator***. Proporciona información de lo que está pasando dentro de una aplicación *Spring Boot* en ejecución.

2.2.2 React.js

React.js [8] es un *framework JavaScript* para la creación de interfaces gráficas de usuario en la web originalmente creado por Facebook. Permite la creación de *single-page application (SPA)* o "aplicaciones de una sola página" con datos que cambian constantemente. *React* facilita la creación de aplicaciones que pueden escalar en el tiempo separando los componentes que forman la página en pequeños componentes independientes que colaboran para conformar la página completa. Así, se consigue que el código de la aplicación sea más fácilmente mantenible.

Las SPAs [9] son aplicaciones compuestas por componentes individuales que pueden ser modificados o reemplazados de manera independiente. Así, no es necesario recargar la página completa con cada interacción del usuario. En una SPA se cargan todos los archivos necesarios, como pueen ser CSS, imágenes y scripts inicialmente al entrar en la página. Luego, se carga de forma dinámica el contenido adecuado dependiendo de las acciones del usuario. Así se consigue una experiencia de usuario más fluida, ya que una vez cargada la página la primera vez, solamente se necesitan recargar partes específicas en lugar de toda la página.

2.3 Herramientas de desarrollo

2.3.1 Eclipse IDE

La plataforma Eclipse [10] está diseñada para crear entornos de desarrollo integrado (IDE) que se pueden usar para crear aplicaciones tan diversas como sitios web, programas JavaTM integrados, programas C++ y Enterprise JavaBeans.

Originalmente fue desarrollado por IBM [11], actualmente es mantenido por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta el código abierto. Inicialmente fue liberado bajo la *Common Public License*, después fue re-licenciado bajo la *Eclipse Public License*.

A diferencia de otros entornos monolíticos donde todas las funcionalidades ya vienen incluidas dentro del producto base, las neceseite el usuario o no, Eclipse permite ampliar sus

capacidades empleando módulos independientes (*plugins*).

En este proyecto se utilizó Eclipse para el desarrollo del servicio hecho en Java.

2.3.2 Sublime Text

Sublime Text [12] es un editor de código fuente y de texto multiplataforma que contiene multitud de características como puede ser resaltado de sintaxis, sangría automática, reconocimiento de tipo de archivo, macros para automatizar tareas repetitivas, etc...

Además, la comunidad ha creado Package Manager para realizar el descubrimiento, la instalación, actualización y la administración de numerosos y prácticos paquetes, extensiones o *plugins*. Desarrollados la mayoría de ellos con licencias de software libre y mantenidos por la propia comunidad. La principal ventaja de Sublime Text, en comparación con un IDE como puede ser Eclipse, es que es más ligero y por este motivo, su ejecución es más fluida en determinados escenarios.

En este proyecto, *Sublime Text*, fue utilizado para el desarrollo de la parte de cliente gráfico con React.

2.3.3 Maven

Maven [13] es una herramienta que permite la automatización de la construcción de proyectos Java. Se centra en dos aspectos, describe cómo se debe construir el software y declara las dependencias que este tiene.

En un fichero XML denominado POM (*Project Object Module*) se describe la forma en la que se debe construir el software, sus dependecias con otros módulos o componentes tanto externos como propios, el orden de construcción, *plugins* necesarios, etc... *Maven* descarga dinámicamente tanto librerías Java como *plugins Maven* de uno o varios repositorios, como por ejemplo el "*Maven Central Repository*".

2.3.4 Git

Git [14] es un sistema de control de versiones distribuido caracterizado principalmente por su velocidad, rendimiento, flexibilidad, y usabilidad. Diseñado por Linus Torvalds (primer prototipo , 7 de Abril de 2005), fue creado inicialmente con el proposito de ser el sistema de control de versiones para el Kernel de Linux, debido a que los sistemas anteriormente utilizados les costaba mucho tiempo manejar un código fuente tan pesado.

2.3.5 SQuirreL SQL Client

SQuirreL SQL Client [15] es un programa con interfaz gráfica Java que te permite ver la estructura de una base de datos compatible con JDBC, consultar datos de sus tablas, lanzar comandos SQL, etc... La versión mínima de Java admitida es 1.8.x a partir de la versión 3.8.1 de SQuirreL. Su funcionalidad se puede extender a través de *plugins*.

2.3.6 Overleaf

Overleaf [16] es una empresa social que construye herramientas colaborativas de creación para científicos. Su principal producto es un editor colaborativo en línea y en tiempo real para trabajos, tesis, informes técnicos y otros documentos escritos en el lenguaje de marcado LaTeX. Con el tiempo se ha convertido en una parte integral, no solo de las colaboraciones en investigaciones, si no también de la ensañanza académica.

El objetivo principal de *Overleaf* es abordar los problemas que surgen a la hora de escribir artículos en colaboración y hacer que el poder LaTeX sea más accesible para todos los científicos y escritores técnicos, en todas las etapas de su carrera.

2.3.7 Postman

Postman [17] es un programa que cuenta con una interfaz gráfica sencilla que permite el diseño, *"mock"* o simulacro, *debug* o depurado, la realización de pruebas, documentación, monitorización y publicación de APIs, todo desde un único sitio. Para este proyecto se ha utilizado *Postman* únicamente para la realiación de pruebas sobre el API del servicio.

2.4 Sistemas de Gestión de Bases de Datos

2.4.1 MySQL

MySQL [18] es un *software* que presenta un servidor robusto, rápido, *multithread* y multiusuario de bases de datos *SQL (Structured Query Language)*. *MySQL* está destinado a sistemas de producción de carga pesada de misión crítica, así como a la incorporación en software de implementación masiva. Es posible utilizar *MySQL* bajo dos licencias distintas, puede ser utilizado como un producto *Open Source* bajo las condiciones de la licencia *GNU (General Public License)*, o se puede adquirir la versión comercial estándar de Oracle.

2.5 APIs

2.5.1 OMDb API

OMDb (*Open Movie Database*) [19] API es un API abierta que permite recuperar información de películas, como puede ser su argumento, reparto, puntuación en IMDb (*Internet Movie Database*), o en Rotten Tomatoes, poster, imágenes, etc...Son los propios usuarios los que proporcionan y mantienen todo el contenido que ofrece el API.

2.6 Otros

2.6.1 HTTP

HTTP (*Hypertext Transfer Protocol*) [20] es un protocolo de nivel de aplicación utilizado por sistemas de información distribuidos, colaborativos e hipermedia. Es un protocolo genérico, sin estado, orientado a objetos que se puede usar para muchas tareas. Es posible expandir su uso a través de la extensión de sus métodos de solicitud (comandos).

2.6.2 JSON

JavaScript Object Notation (JSON) [21], es un formato de intercambio de datos ligero, basado en texto e independiente del lenguaje. JSON define un pequeño conjunto de reglas de formato para la representación de datos estructurados. Se deriva de los literales de objetos de JavaScript, como se define en la terecera edición del estándar de lenguaje de programación ECMAScript.

JSON puede representar cuatro tipos primitivos (cadenas, números, valores booleanos y nulos) y dos tipos estructurados (objetos y matrices).

2.6.3 REST

Representational State Transfer (REST) [22] es un estilo de arquitectura Web con una restricciones específicas definidas por Roy Fielding en su tésis doctoral.

El estilo de arquitectura REST es comunmente utilizada para construir APIs de servicios Web modernos. Cuando una API Web cumple con las reglas de arquitectura definidas por REST se llama *API REST*. En general, una API REST consiste en un conjunto de recursos interconectados, sobre los cuales se pueden hacer diferentes operaciones estándar.

Capítulo 3

Análisis de viabilidad

En este capítulo se expondrá en análisis de viabilidad llevado a cabo antes del comienzo de la realización del mismo. Realizar un análisis de viabilidad es esencial para cualquier tipo de proyecto, este sirve para poder determinar si un proyecto se puede llegar a finalizar con éxito. El análisis de viabilidad se ha dividido en tres apartados: económico, técnico y de mercado.

3.1 Viabilidad económica

Inicialmente se ha llevado a cabo el análisis de la viabilidad económica del proyecto. Se ha realizado en base a los tres principales costes que tiene un proyecto *software*: costes en recursos humanos, recursos *software* y recursos *hardware*.

En cuanto a los recursos humanos, al ser solamente una persona la que se va a encargar de todas las fases de desarrollo no es posible decir que supone un gran coste para el proyecto.

En relación con los recursos *software*, se intentará en la medida de lo posible utilizar *software open source* para minimizar los costes. Al ser una aplicación web el objetivo de este proyecto, no existe ningún inconveniente para encontrar el *software open source* adecuado para su realización, ya que existe multitud de opciones e información que se encuentra fácilmente en la web.

En lo relativo al *hardware*, solamente es necesario un ordenador portátil de gama media y una conexión a internet. Por este motivo y los anteriores se puede decir que el proyecto es viable económicamente, ya que la inversión necesaria no es elevada.

3.2 Viabilidad técnica

Como se ha visto en el capítulo 2, las tecnologías y herramientas utilizadas son ampliamente empleadas y reconocidas dentro del ámbito del desarrollo *software*, por lo que suponen un riesgo bajo. Además, el conocimiento previo de la mayoría de ellas por parte del desarrollador hace que disminuya más el riesgo asumido. Por estos motivos se puede afirmar que el proyecto es viable técnicamente.

3.3 Viabilidad de mercado

En la actualidad, ya existen numerosas opciones relacionadas con el ámbito de las películas, por ejemplo *FilmAffinity*, *IMDb* o *Rotten Tomattoes*. Pero todas estas aplicaciones web están más orientadas hacia constituir una base de datos de películas que se pueda consultar de forma rápida y sencilla. Mientras que la aplicación de este proyecto, tiene la intención de ser más parecida a una red social motivada por el gusto de sus usuarios por las películas. Incentivando a estos a seguir a otros usuarios con gustos similares, recomendar películas y compartir sus gustos y opiniones con sus amigos en la Web.

Por este motivo, se puede afirmar que este proyecto tiene un hueco, aunque sea pequeño, en el mercado actual, por lo que es viable su desarrollo.

Capítulo 4

Introducción al desarrollo realizado

En este capítulo de la memoria se explicará la arquitectura global del sistema, así como la metodología empleada para su desarrollo.

4.1 Arquitectura global del sistema

El sistema se ha creado siguiendo patrón de diseño *Model-View-Controller (MVC)* [23]. Este patrón de diseño es altamente utilizado en aplicaciones interactivas, en las que el usuario interactua con el sistema y este responde de alguna forma. La idea clave de este patrón de diseño es la de separar (o desacoplar) en capas lógicas el código de la aplicación. Esto permite aislar las partes de la aplicación que realizan acciones similares. Con esto, se consigue que el software resultante sea más mantenible y escalable

Se cuenta por un lado con el módulo **Vista** o ***View*** que se encarga de presentarle al usuario final la información y responder a sus acciones. La parte **Controlador** o ***Controller*** que se encarga de procesar las acciones originadas por la interacción del usuario con el sistema. Esta capa controladora se encuentra en el medio y tiene como objetivo principal comunicar las otras dos. El **Modelo** o ***Model*** es la parte de la aplicación que contiene tanto la información representada por la Vista como la lógica que cambia esta información en respuesta a la interacción del usuario.

Como se puede ver en la figura 4.1, dentro de cada una de las partes del patrón *MVC* se ha dividido el código de la aplicación en más grupos lógicos. En cada uno de ellos se utilizan diferentes tecnologías, lenguajes y *frameworks* de programación. A continuación se explicará brevemente cada una de las partes:

- **Capa de acceso a datos.** Es la capa lógica de código que se encarga de acceder a la base de datos subyacente y recuperar los datos necesarios para la capa de lógica de negocio.

Esta capa está programada en Java y se ha utilizado la tecnología Spring Data JPA (*Java Persistence API*), la cual permite mapear objetos Java a entidades de Base de Datos y realizar operaciones sobre ellos a través de la implementación de los repositorios.

- **Capa lógica de negocio.** Es la capa en la que se implementa la lógica interna de los casos de uso de la aplicación. Dentro de esta capa, que se ha desarrollado también en Java apoyándose en Spring para el manejo de la inyección de dependencias y la transaccionalidad, se hace uso de los repositorios propios de la capa de acceso a datos.

- **Servicio REST.** Esta capa se encarga de comunicar la vista con la capa de lógica de negocio. Recibe peticiones en forma de solicitudes HTTP desde la vista, las resuelve apoyándose en la capa de lógica de negocio y devuelve el resultado de nuevo a la vista. Esta capa está desarrollada en Java utilizando el módulo Web MVC del framework Spring para la anotación de los métodos que actuarán como los manejadores de las distintas operaciones definidas y para el mapeo de las entidades de respuesta a formato JSON.

- **Vista.** En esta capa se encierra toda la lógica de la interfaz de usuario de la aplicación. En general, responde a las acciones del usuario pidiendo los datos necesarios realizando llamadas al servicio REST y se los presenta al usuario. Esta desarrollada utilizando el lenguaje JavaScript y el framework React. En cuanto a los estilos se ha utilizado Ant Design para darle un mejor aspecto sin incrementar demasiado el coste.

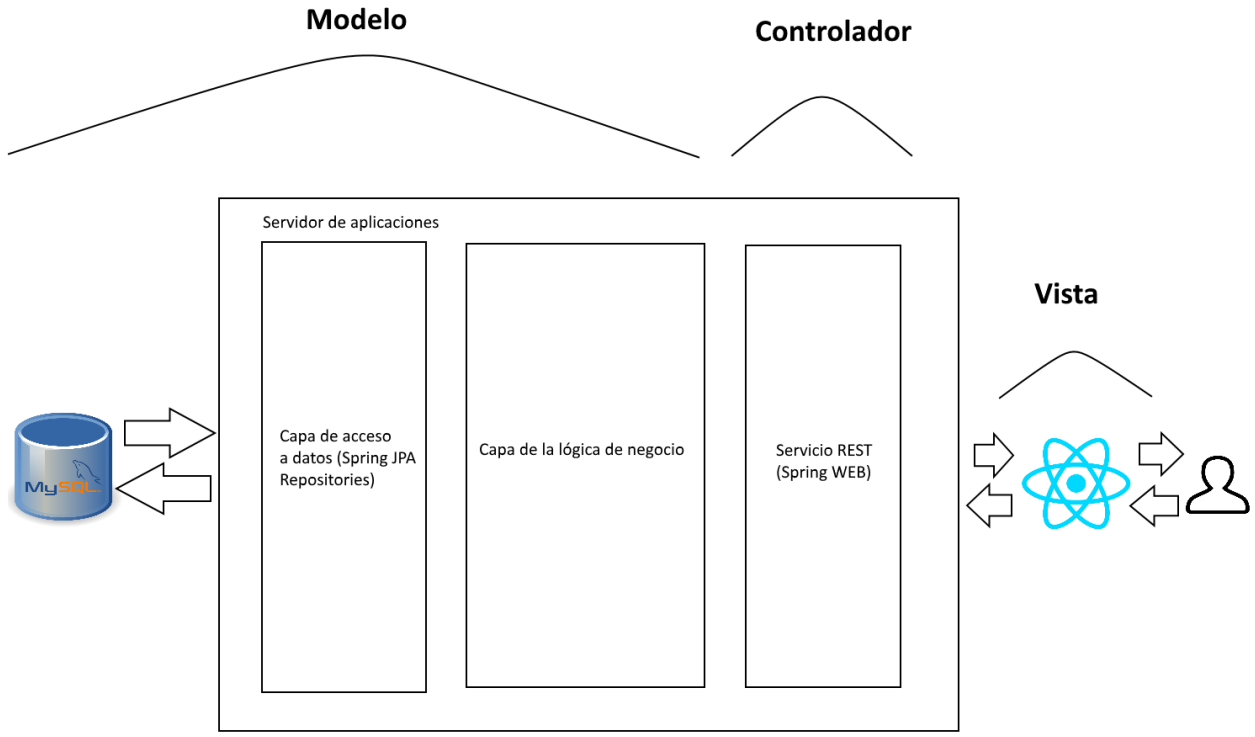


Figura 4.1: Arquitectura General del Sistema

4.2 Metodología utilizada

Para la elaboración del presente proyecto *software* se ha escogido una versión más específica, simplificada y en general más utilizada del Proceso Unificado de Desarrollo Software denomimada *Rational Unified Process* (RUP) [24].

4.2.1 Rational Unified Process

El *Rational Unified Process* o RUP es un marco de procesos de ingeniería de software desarrollado y comercializado por Rational Software. Está compuesto por la acumulación de conocimiento y buenas prácticas aportadas por numerosos colaboradores durante muchos años, en

una amplia variedad de situaciones. Al aplicar este proceso se puede producir software de alta calidad que satisfaga las necesidades de los usuarios finales, y hacer dentro de un presupuesto y fechas predecibles.

Esta metodología tiene un enfoque de desarrollo de software iterativo, centrado en la arquitectura y basado en casos de uso.

Como se ha dicho RUP utiliza un **enfoque iterativo**, es decir, una secuencia de pasos incrementales o iteraciones. En cada una de estas iteraciones se llevan a cabo todas, o la mayoría de las disciplinas de desarrollo software (análisis de requisitos, diseño, implementación y pruebas) para producir una implementación parcial del sistema final. Cada iteración sucesiva se basa en el trabajo de iteraciones anteriores para refinar el sistema hasta que el producto final esté completo.

Es importante establecer las bases de la **arquitectura** que va a tener el software que se va a producir, ya que esto permite minimizar considerablemente los riesgos inherentes a ella. Además, establecer una arquitectura estable desde el principio también facilita la comunicación y localiza el impacto de los cambios.

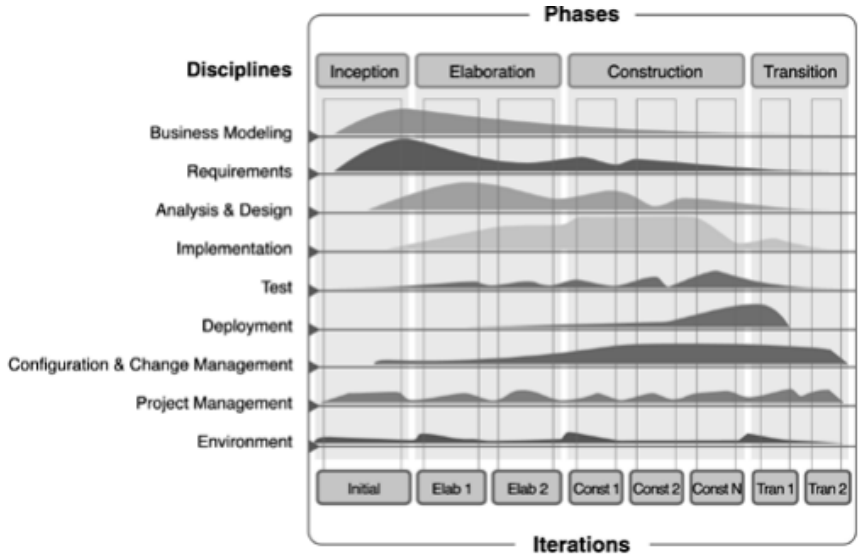


Figura 4.2: Fases Rational Unified Process

Como se puede ver en la figura 4.2, el *Rational Unified Process* se organiza en dos dimensiones:

- La dimensión horizontal representa la estructura dinámica o dimensión temporal del

proceso. Muestra cómo se desarrolla el proceso, expresado en términos de ciclos, fases e iteraciones.

- La dimensión vertical representa la estructura estática del proceso. Describe cómo los elementos de proceso (actividades, disciplinas, artefactos y roles) se agrupan lógicamente en flujos de trabajo.

Dentro de la dimensión horizontal se pueden diferenciar cuatro fases distintas con un objetivo bien marcado:

- **Comienzo**. En esta fase se establecen los requisitos de alto nivel y el alcance del proyecto. También es en la fase en la que todas las partes se pone de acuerdo para decidir si se pone en marcha o no el proyecto.
- **Elaboración.** En esta fase se analiza el dominio del problema, se implementan los componentes que conforman el núcleo del sistema. En esta fase también se establece una arquitectura ejecutable y estable teniendo en cuenta el alcance de todo el proyecto, así se consiguen mitigar los principales riesgos técnicos.
- **Construcción.** Fase en la que se realiza la mayor parte de la implementación para pasar de una arquitectura ejecutable a la primera versión operativa del sistema. En esta fase se hace uso de la mayor parte de los recursos, para conseguir los objetivos establecidos en las primeras fases. La fase con la implementación de una versión beta completamente funcional del sistema, aunque es posible que contenga ciertos defectos o errores que se corregirán en la última fase.
- **Transición**. Esta fase consiste en la aseguración de que el software cubra las necesidades de sus usuarios. Esto incluye probar el producto para el lanzamiento y realizar ajustes menores basados en los comentarios de los usuarios, como pueden ser la configuración, la instalación y los problemas de uso.

4.2.2 Iteraciones

Empleando la metodología descrita anteriormente se ha conseguido dividir el proyecto en 12 iteraciones.

En cada una de estas iteraciones se han realizado las fases de análisis, diseño, implementación y pruebas. Cada una de ellas incorpora nueva funcionalidad a la anterior. A continuación se explicaran brevemente.

1. **Iteración 1: Definición del proyecto.**
   En esta fase del proyecto se ha llevado a cabo el análisis de viabilidad y la definición

del dominio del proyecto. Además, también se ha llevado a cabo un estudio sobre las tecnologías disponibles para determinar cuáles se iban a utilizar intetando minimizar los riesgos inherentes al desconocimiento de dichas tecnologías para el proyecto.

Durante esta fase también se han captado los requisitos y definido de manera menos formal los casos de uso y la estructura globlal que tendrá el sistema final de este proyecto.

2. **Iteración 2: Puesta en marcha del proyecto.**
   Después de realizar la definición general del proyecto, se han formalizado, refinado y detallado por escrito todos los casos de uso. Otra de las tareas que pertenece a esta iteración es la de la construcción del entorno en el que se dearrollará la aplicación. Al finalizar esta fase se terminado con un arquetipo con las dependencias esenciales definidas, instaladas y funcionando. Además de todas las herramientas necesarias.

3. **Iteración 3: Gestión de usuarios.**
   En esta tercera iteración se ha llevado a cabo la generación del subsitema que gestiona los usuarios. Se han creado las entidades persistentes y todas las operaciones necesarias sobre ellas. Se han realizado las fases de análisis, diseño, implementación y pruebas sobre una primera versión en la que se permite el acceso y registro de usuarios. También se ha generado el módulo de administración de usuarios para que sea utilizado por los usuarios adminstradores. En él se permite añadir, eliminar y modificar usuarios.

4. **Iteración 4: Perfil de usuario.**
   En esta iteración se ha trabajado en el desarrollo del perfil de los usuarios que pueden ver otros usuarios. En este perfil se permite visualizar la información básica de los usuarios además de su actividad reciente clasificada en tres tipos distintos: recomendación de películas, puntuación de películas y usuarios seguidos. En la pantalla de perfil de usuario también se permite seguir a otro usuario para poder conocer su actividad reciente y poder recomendarle películas. Ambas acciones se implementarán en futuras iteraciones.

5. **Iteración 5: Gestión de películas.**
   En esta iteración se ha realizado la construcción de los módulos relacionados con la gestión de películas. Se han realizado las cuatro fases sobre la versión anterior para terminar con una nueva versión que permite visualizar la información básica sobre películas a los usuarios registrados. También se ha incorporado al módulo de administración la posibilidad de añadir, modificar y eliminar películas para los usuarios registrados como administradores. Además, en esta iteración se ha añadido un buscador por título para poder encontrar las películas almacenadas en el sistema.

6. **Iteración 6: Gestión de puntuaciones.**
   En esta iteración se ha trabajado para incorporar la funcionalidad que permite puntuar una película y a la vez visualizar la puntuación media que ha obtenido una película. Como en las anteriores iteraciones se han realizado las cuatro fases del proceso de desarrollo para llevarla a cabo.

7. **Iteración 7: Gestión de críticas y recomendación de películas.**
   En esta iteración se ha llevado a cabo el desarrollo de toda la funcionalidad referente a la publicación, edición y consulta de críticas de películas hechas por usuarios con el rol de crítico registrado en el sistema. En esta iteración también se ha realizado el proceso software para el desarrollo de la funcionalidad que permite a un usuario registrado recomendar una película concreta a alguno o algunos de los usuarios a los que sigue.

8. **Iteración 8: Integración con OMDb API.**
   En la séptima iteración se ha realizado el análisis, diseño e implementación de la integración de la aplicación de este proyecto con la API pública de consulta de información de películas denominada OMDb API. Esta integración permite mostrar la puntuación que tiene la película que se está consultando en otras aplicaciones conocidas como por ejemplo Rotten Tomatoes.

9. **Iteración 9: Actividades recientes de usuarios seguidos en la pantalla de inicio.**
   En esta novena iteración se ha comenzado a montar la pantalla de inicio de la aplicación. Se ha empezado con la parte de la pantalla en la que se muestran las actividades recientes de los otros usuarios seguidos por el propio usuario, motrando primero las más recientes y luego las más antiguas. Diferenciando entre los tres tipos ya nombrados en la iteración 4.

10. **Iteración 10: Películas más vistas y mejor valoradas en la pantalla de inicio.**
    Después de terminar la iteración anterior con una primera versión de la pantalla de inicio de la aplicación, en esta décima iteración se le van a añadir dos elementos más. Una parte en la que se verán las películas más vistas, es decir, las que más se han votado ordenadas descedentemente por este criterio. Y otra parte en la que se mostrarán las películas que tienen mayor puntuación media, es decir, las mejor valoradas por los usuarios.

11. **Iteración 11: Buscador de películas.**
    Partiendo del producto resultante de la iteración anterior, se va a proceder a crear un buscador de películas que permita filtrar por varios campos. A este buscador se va a poder acceder desde los botones de ”Mostrar más” de los dos paneles construidos en la iteración anterior. Dependiendo del panel desde el que se acceda al buscador, las

películas se mostrarán ordenadas de las más vistas a las menos vistas, o ordenadas de las mejores puntuadas a las peores. Y con esta iteración se da por finalizado el desarrollo de la aplicación.

12. **Iteración 12: Elaboración de la memoria.**
    En esta iteración final se va a realizar la elaboración de la memoria del trabajo realizado para el pryecto. Se ha realizado en LaTeX debido a su potencia y eficacia a la hora de elaborar documentos académicos.

Capítulo 5

Planificación y análisis de costes

Para la realización de cualquier proyecto de desarrollo software hay que tener en cuenta dos elementos fundamentales, el tiempo y el coste. Un jefe de proyecto debe conocer estos dos parámetros previamente antes de comenzar con el proyecto, y para poder conocerlos es necesario llevar a cabo una planificación y un análisis de costes. Al hacer esto, es posible minimizar ambos y asi conseguir un resultado óptimo y predecible.

5.1 Recursos

Para la realización de este proyecto han sido necesarios dos tipos de recurso: recursos humanos y técnicos.

5.1.1 Recursos humanos

Para la realización del proyecto se han simulado varios perfiles de recursos humanos. Sin embargo, detrás de todos estos perfiles se encuentra el autor de este pryecto en colaboración con el director del proyecto. Los perfiles son los siguientes:

- **Jefe de proyecto:** es el encargado de la coordinación y gestión de todos los recursos imbolucrados en el proyecto con el objetivo de que este se desarrolle correctamente. También se encarga de definir los requisitos del proyecto y realizar la planificación y el seguimiento.
- **Analista:** es el encargado de capturar los requisitos establecidos por el jefe de proyecto, además de realizar el análisis técnico y funcional. En las fases de desarrollo propiamente dichas, el analista se encarga de asegurar que los requisitos del cliente se cumplan y también de resolver cualquier duda que les surga a los programadores.

- **Programador:** utilizando los documentos funcionales y técnicos generados por el analista realiza la implementación de esos requisitos en la aplicación. También se encarga de realizar las pruebas.

5.1.2 Recursos técnicos

Los recursos técnicos necesarios para el desarrollo de este proyecto han sido un ordenador portátil con las siguientes especificaciones:

- **Marca y modelo:** Sony Vaio SVF152A29M
- **Procesador:** Intel Core i5-3337U CPU @ 1.8GHz
- **Memoria RAM:** 6.00 GB DDR3

En este ordenador se ha instalado todo el *software* y las herramientas necesarias para el desarrollo del proyecto ya comentadas antes.

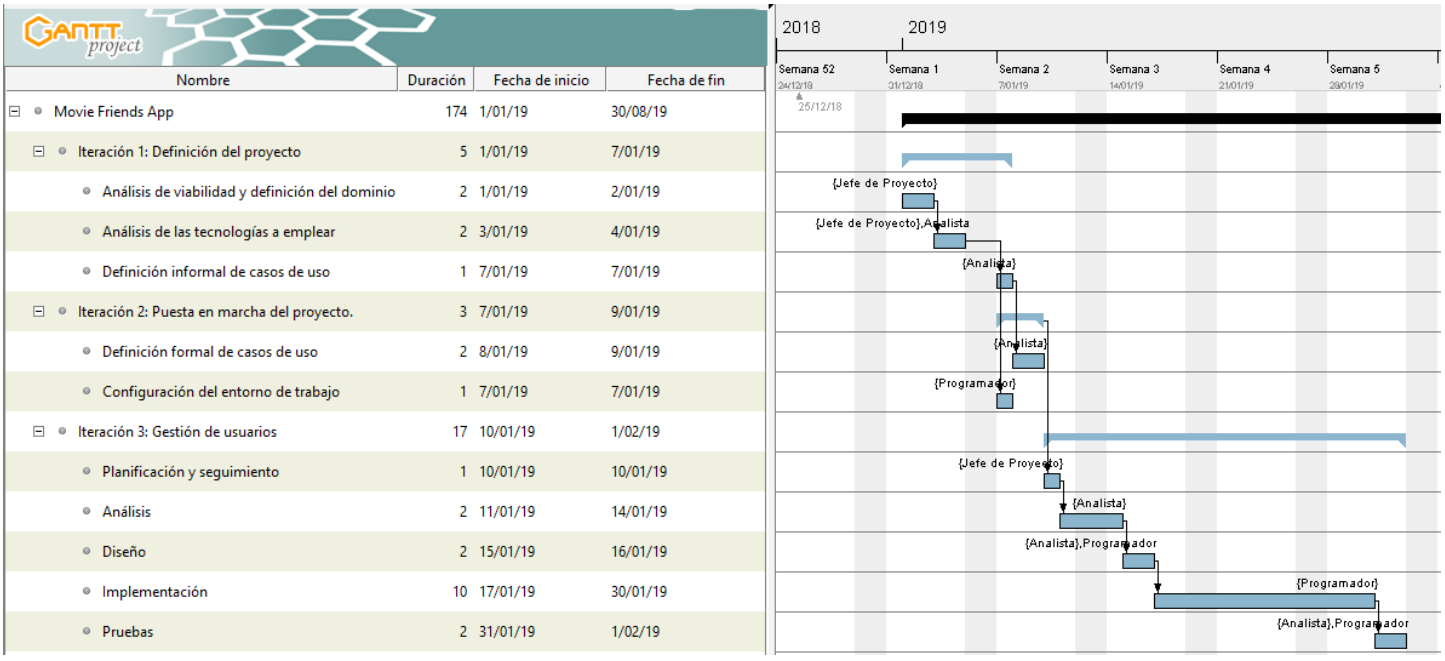
5.2 Panificación y costes.

Las iteraciones definidas en el apartado 4.2.2 son las que se han llevado a cabo para realizar el proyecto. Estas iteraciones se han dividido en tareas más pequeñas, se han planificado y estimado para determinar el tiempo y el coste que implica su desarrollo.

Para determinar el coste de cada tarea es necesario conocer el coste individualde cada recurso:

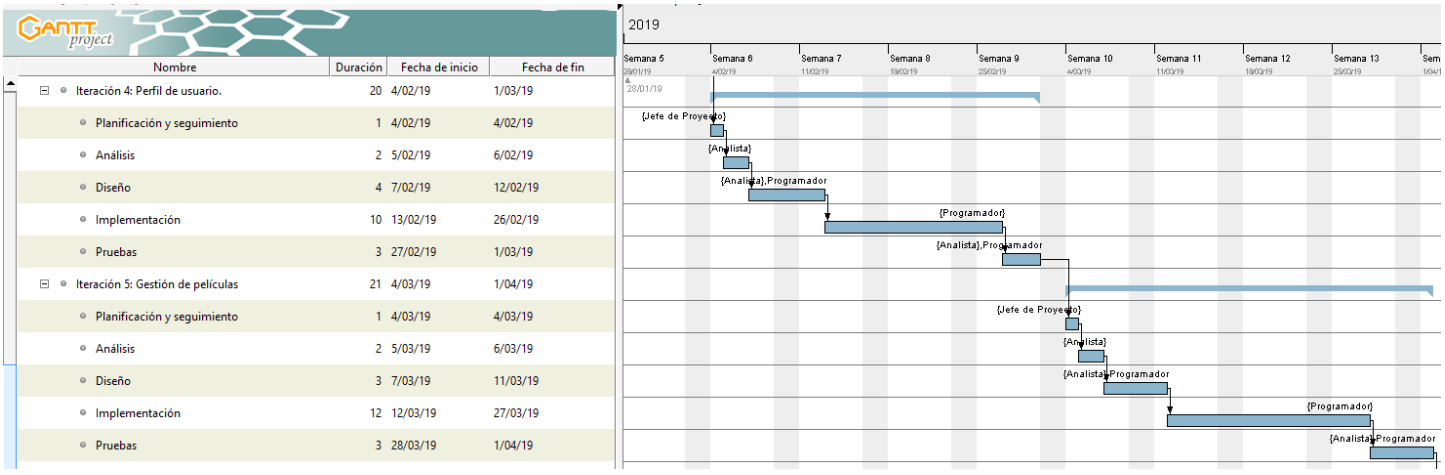
Recurso	Coste
Jefe de proyecto	30€/h
Analista	20€/h
Programador	15€/h
Ordenador portátil	600€

Una vez se conocen los costes individuales de cada recurso se va a proceder a realizar un diagrama de Gantt, para ilustrar de forma gráfica la duración del proyecto, separando las iteraciones en pequeñas tareas las cuales tendrán recursos asignados, fecha de inicio y fecha de fin y tendrán dependencias entre si. A continuación en las figuras 5.1, 5.2, 5.3, 5.4 y 5.5 se pueden ver todas las iteraciones desglosadas en tareas.



Nombre	Duración	Fecha de inicio	Fecha de fin
Movie Friends App	174	1/01/19	30/08/19
Iteración 1: Definición del proyecto	5	1/01/19	7/01/19
Análisis de viabilidad y definición del dominio	2	1/01/19	2/01/19
Análisis de las tecnologías a emplear	2	3/01/19	4/01/19
Definición informal de casos de uso	1	7/01/19	7/01/19
Iteración 2: Puesta en marcha del proyecto.	3	7/01/19	9/01/19
Definición formal de casos de uso	2	8/01/19	9/01/19
Configuración del entorno de trabajo	1	7/01/19	7/01/19
Iteración 3: Gestión de usuarios	17	10/01/19	1/02/19
Planificación y seguimiento	1	10/01/19	10/01/19
Análisis	2	11/01/19	14/01/19
Diseño	2	15/01/19	16/01/19
Implementación	10	17/01/19	30/01/19
Pruebas	2	31/01/19	1/02/19

Figura 5.1: Diagrama de Gantt - Detalle iteraciones 1



Nombre	Duración	Fecha de inicio	Fecha de fin
Iteración 4: Perfil de usuario.	20	4/02/19	1/03/19
Planificación y seguimiento	1	4/02/19	4/02/19
Análisis	2	5/02/19	6/02/19
Diseño	4	7/02/19	12/02/19
Implementación	10	13/02/19	26/02/19
Pruebas	3	27/02/19	1/03/19
Iteración 5: Gestión de películas	21	4/03/19	1/04/19
Planificación y seguimiento	1	4/03/19	4/03/19
Análisis	2	5/03/19	6/03/19
Diseño	3	7/03/19	11/03/19
Implementación	12	12/03/19	27/03/19
Pruebas	3	28/03/19	1/04/19

Figura 5.2: Diagrama de Gantt - Detalle iteraciones 2

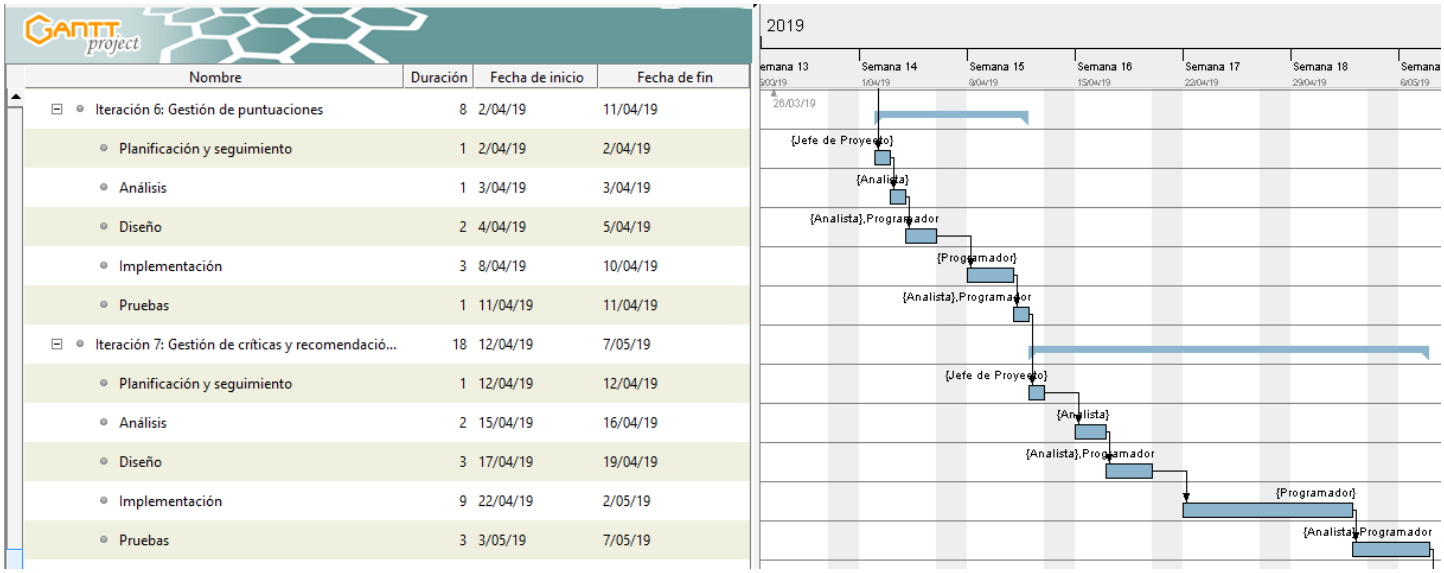


Figura 5.3: Diagrama de Gantt - Detalle iteraciones 3

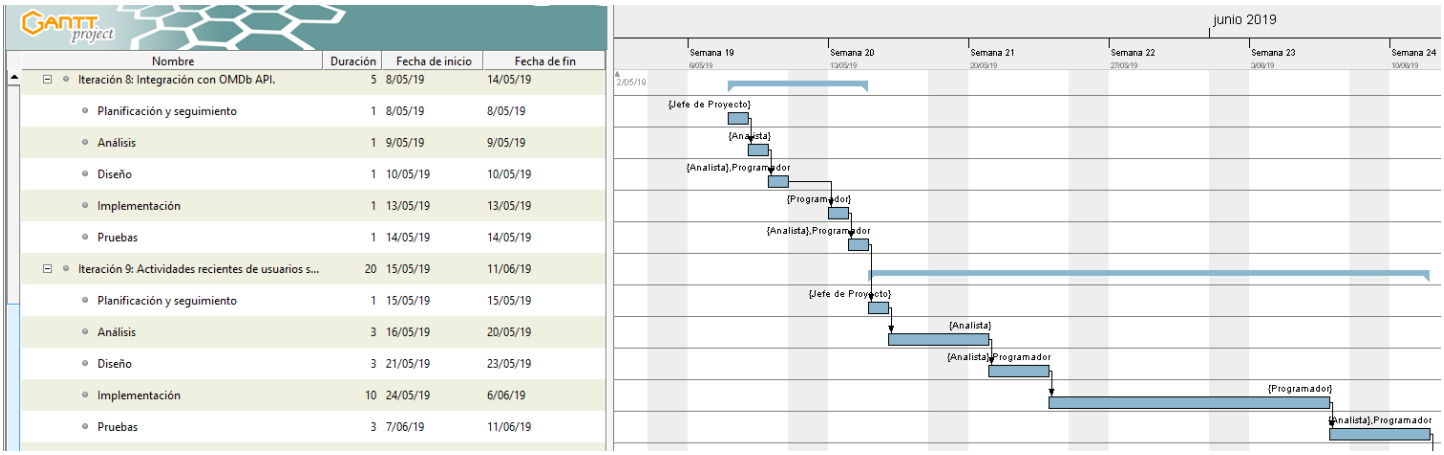


Figura 5.4: Diagrama de Gantt - Detalle iteraciones 4

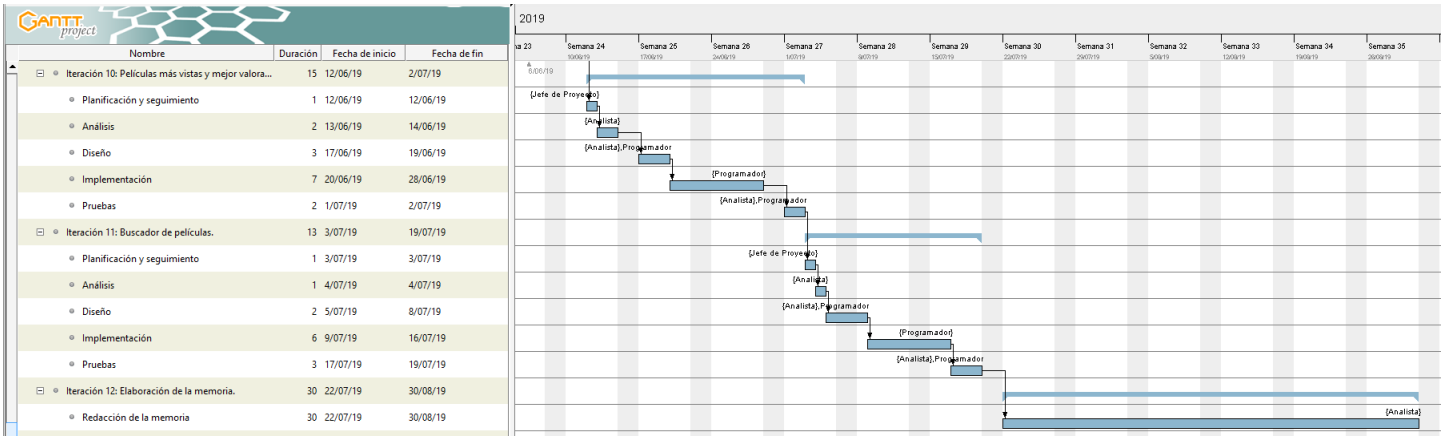


Figura 5.5: Diagrama de Gantt - Detalle iteraciones 5

En la figura 5.6 se puede ver el resumen por iteración. Según la planificación realizada, el desarrollo del proyecto supondrá un total de **174 días de duración**, dedicándole aproximadamente 4 horas de trabajo por día. El proyecto comienza el día **1 de enero de 2019** y termina el **30 de agosto de 2019**.

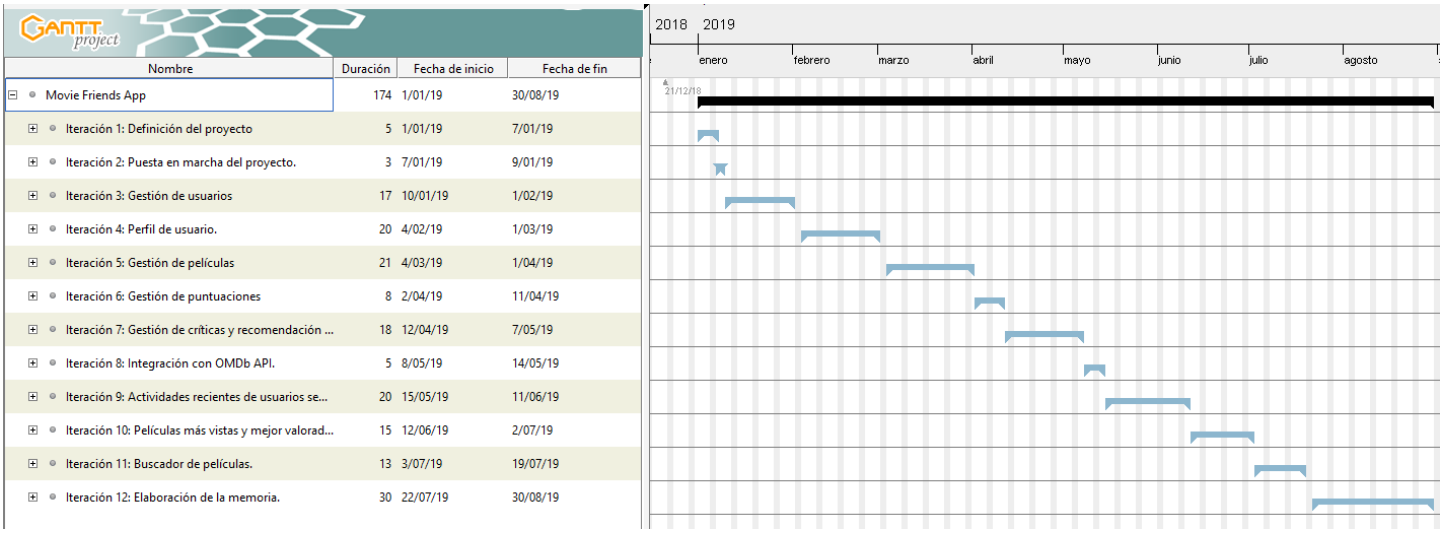


Figura 5.6: Diagrama de Gantt - Iteraciones

A continuación se realizará el cálculo del coste estimado del desarrollo del proyecto, teniendo en cuenta tanto los recursos técnicos como los recursos humanos:

- **Recursos humanos:** El coste de los recursos humanos se calcula teniendo en cuenta el tiempo empleado por cada recurso al proyecto. En total el proyecto ha supuesto un total de **872 horas de trabajo**, las cuales se reparte de la siguiente forma:

Jefe de proyecto: 52 horas que equivalen a 1.560€

Analista: 368 horas que equivalen a 7.360€

Programador: 452 horas que equivalen a 6.780€

- **Recursos técnicos:** los recursos técnicos se han dividido en dos, recursos *software* y recursos *hardware*:

  Recursos hardware: solamente un ordenador portátil que ha costado aproximadamente 600€

  Recursos software: debido a que se han utilizado tecnologías de *software* libre el coste es de 0€.

En la siguiente tabla se puede ver el coste total estimado del proyecto:

Concepto	Coste
Recursos humanos	15.700,00 €
Jefe de proyecto	1.560,00 €
Analista	7.360,00 €
Programador	6.780,00 €
Recursos técnicos	600,00 €
Software	0,00 €
Hardware	600,00 €
Subtotal	16.300,00 €
IVA 21%	3.423,00 €
Total	**19.723,00 €**

Capítulo 6

Requisitos del sistema

En este capítulo se expondrán los requisitos que deberá cumplir el sistema final. Se definirán también los actores que interactuarán con el aplicativo haciendo uso de las distintas funcionalidades detalladas en los casos de uso.

6.1 Especificación de requisitos

El principal objetivo de la especificación de requisitos *software* [25] es el de exponer de forma clara, concisa y sin ambigüedades las necesidades de los usuarios finales del sistema. Estos pueden dividirse en 2 categorías: requisitos funcionales y requisitos no funcionales.

6.1.1 Requisitos funcionales

Los requisitos funcionales son los que definen las funciones que el sistema realizará, describen las transformaciones que el sistema realiza sobre las entradas para producir salidas. Es importante que se describa el ¿Qué? y no el ¿Cómo? se deben hacer esas transformaciones [25].

A continuación se explicarán brevemente los requisitos fundamentales, los demás se expondrán de forma más detallada en la sección 6.3, donde se definen todos los casos de uso.

El acceso a la aplicación estará **restringido a los usuarios con cuenta registrada**. Estos usuarios serán de tres tipos distintos, los **usuarios administradores** que podrán realizar todas las operaciones además de poder administrar las entidades persistentes del sistema. Los **usuarios básicos** que tendrán un acceso limitado, podrán visualizar, puntuar y recomendar películas. Y por últimos, los **usuarios de tipo críticos**, podrán realizar las misma operaciones que los usuarios básicos pero con la excepción de que también podrán crear, modificar y eliminar críticas de películas.

6.1.2 Requisitos no funcionales

Los requerimientos no funcionales, por otra parte, tienen que ver con características que limitan el sistema, como por ejemplo, el rendimiento, interfaces de usuario, fiabilidad, seguridad, portabilidad, etc. [25]

Para este proyecto se han acordado los siguientes requisitos no funcionales:

- **Seguridad**. Se controla el acceso a la aplicación mediante la autenticación de los usuarios cuyas contraseñas se guardan encriptadas en base de datos. Dependiendo del tipo de usuario, se le limita las operaciones que puede realizar a través de roles de usuario.
- **Fiabilidad**. Todos los errores que se puedan producir en tiempo de ejecución están controlados para poder dar una respuesta adecuada al usuario en caso de que ocurran.
- **Usabilidad y simplicidad.** El uso de la aplicación deberá ser simple e intuitivo. El sistema final será del tipo SPA (*Simple-page Application*), eliminando así muchas interrupciones en la experiencia de usuario haciéndola más fluida.

6.2 Actores

Como se puede ver en la figura 6.1, en la aplicación existen tres tipos de usuario. A continuación se explicarán brevemente qué funciones se le permite realizar a cada tipo.

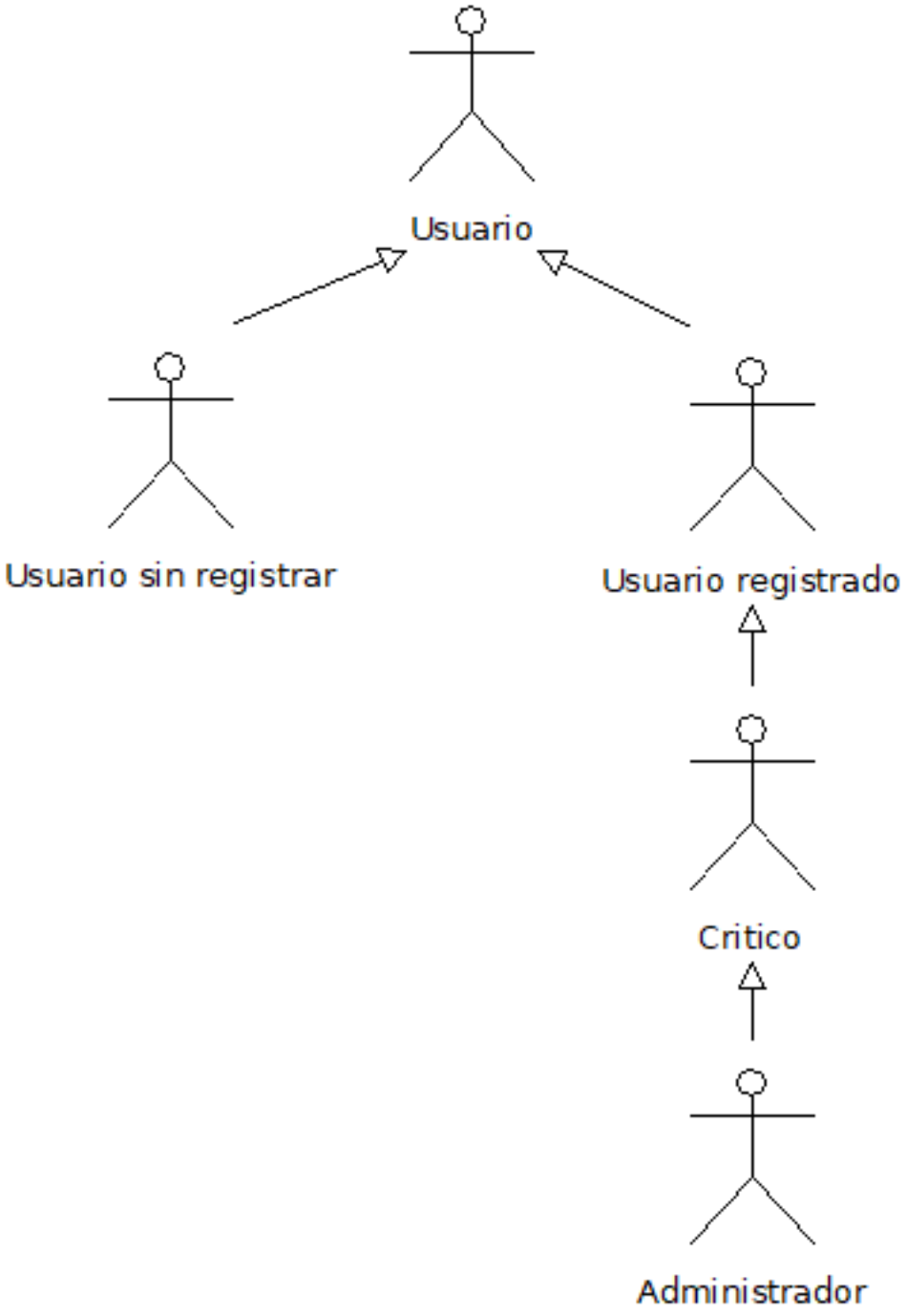


Figura 6.1: Diagrama de actores

- **Usuario sin registrar**. Es el usuario que entra a la aplicación sin estar autenticado en la aplicación. Las operaciones que pueden realizar están muy limitadas, solamente podrá

realizar aquellas que le premitan autenticarse en el sistema.

- **Usuario registrado**. Este tipo de usuario es el usuario base. Se le permite buscar, consultar, puntuar y recomendar películas.
- **Critico**. A este tipo de usuario se le permite realizar las mismas acciones que a un usuario básico pero también podrán crear y modificar críticas de películas.
- **Administrador.** Este tipo de usuario puede realizar todas las operaciones del sistema y además puede administrar las películas y usuarios registrados en el sistema y añadir más.

6.3 Casos de uso

6.3.1 Casos de uso de usuarios

CU-101: Registrar usuario	
Descripción	Un usuario nuevo se registra en el sistema.
Actores	Usuario sin registrar.
Precondiciones	El nuevo usuario no pueda estar registrado en el sistema.
Flujo básico	1.- El usuario pulsa en el botón registrarse situado en la barra superior en la página de inicio de sesión. 2.- Se le muestran al usuario todos los campos que debe cubrir. 3.- El usuario cubre todos los campos necesarios para el registro. 4.- El usuario pulsa el botón ”Registrar” 5.- El sistema valida todos los campos introducidos por el usuario y los almacena en la base de datos. 6.- Se redirige al nuevo usuario registrado a la página principal de la aplicación.
Flujo alternativo	1.- El usuario introduce un dato no válido, en blanco o el email o nombre de usuario ya está registrado en el sistema. 2.- Se informa al usuario de qué datos son incorrectos y cómo debe proceder para solucionarlo.
Postcondiciones	El usuario queda registrado en el sistema, además queda autenticado en la sesión actual.

Cuadro 6.1: CU-101: Registrar usuario

CU-102: Iniciar sesión	
Descripción	Un usuario inicia sesión en el sistema.
Actores	Usuario sin registrar.
Precondiciones	El usuario no puede estar autenticado en el sistema.
Flujo básico	1.- El sistema muestra en su pantalla de inicio ”Nombre de usuario o Email” y contraseña. 2.- El usuario cubre los datos correctamente y pulsa el botón ”Login”. 3.- El sistema valida que los datos sean correctos. Se autentica al usuario y se le redirige a la pantalla principal.
Flujo alternativo	1.- El usuario introduce un dato no válido o en blanco. 2.- Se informa al usuario de qué datos son incorrectos y cómo debe proceder para solucionarlo.
Postcondiciones	El usuario queda autenticado en el sistema en la sesión actual.

Cuadro 6.2: CU-102: Iniciar sesión

CU-103: Cerrar sesión	
Descripción	Un usuario cierra una sesión abierta en el sistema.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono de perfil de usuario y este despliega un menú hacia abajo con varias opciones. 2.- El usuario hace click en la opción ”Logout”. 3.- El sistema cierra la sesión activa del usuario, se le redirige a la pantalla de inicio de sesión y se le muestra un mensaje indicándole que se ha cerrado su sesión.
Flujo alternativo	
Postcondiciones	La sesión activa del usuario queda cerrada en el sistema.

Cuadro 6.3: CU-103: Cerrar sesión

CU-104: Ver perfil de usuario	
Descripción	Un usuario entra en la pantalla de su perfil para consultar sus datos.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono de perfil de usuario y este despliega un menú hacia abajo con varias opciones. 2.- El usuario hace click en la opción ”Profile”. 3.- Se redirige al usuario a la página de su perfil donde puede consultar sus datos.
Flujo alternativo	
Postcondiciones	El usuario verá en pantalla sus datos.

Cuadro 6.4: CU-104: Ver perfil de usuario

CU-105: Crear usuario	
Descripción	Un usuario administrador crea un usuario.
Actores	Administrador.
Precondiciones	El usuario administrador debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono con forma de llave inglesa y este le lleva a la página de administración. Este botón solamente es visible por los usuarios administradores. 2.- El usuario hace click en la pestaña ”Usuarios” y se le despliega un buscador en el que se puede filtrar por nombre completo y por nombre de usuario. 3.- El usuario hace click en el botón ”Agregar usuario”. Se le abre una ventana con todos los datos de usuario necesarios. 4.- El usuario rellena los campos correctamente y pulsa el botón ”Guardar”. El sistema valida todos los campos introducidos por el usuario y los almacena en la base de datos. 5.- Se cierra la ventana modal.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario administrador introduce un dato no válido, en blanco o el email o nombre de usuario ya está registrado en el sistema. 2.- Se informa al usuario administrador de qué datos son incorrectos y cómo debe proceder para solucionarlo. Flujo alternativo 2: 1.- El usuario administrador pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se almacenan los datos introducidos.
Postcondiciones	El usuario queda registrado en el sistema.

Cuadro 6.5: CU-105: Crear usuario

CU-106: Modificar usuario	
Descripción	Un usuario administrador edita un usuario.
Actores	Administrador.
Precondiciones	El usuario administrador debe tener una sesión abierta en el sistema y el usuario que quiere modificar debe existir en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono con forma de llave inglesa y este le lleva a la página de administración. Este botón solamente es visible por los usuarios administradores. 2.- El usuario hace click en la pestaña ”Usuarios” y se le despliega un buscador en el que se puede filtrar por nombre completo y por nombre de usuario. 3.- El usuario hace click en el botón ”editar” del usuario que quiere modificar. Se le abre una ventana modal igual a la que añade un usuario pero con los datos cubiertos. 4.- El usuario modifica los campos que desea correctamente y pulsa el botón ”Guardar”. El sistema valida todos los campos introducidos por el usuario y los actualiza en la base de datos. 5.- Se cierra la ventana modal.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario administrador introduce un dato no válido, en blanco o el email o nombre de usuario ya está registrado en el sistema. 2.- Se informa al usuario administrador de qué datos son incorrectos y cómo debe proceder para solucionarlo. Flujo alternativo 2: 1.- El usuario administrador pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se modifica el usuario seleccionado.
Postcondiciones	Los cambios realizados sobre el usuario quedan almacenados en el sistema.

Cuadro 6.6: CU-106: Modificar usuario

CU-107: Eliminar usuario	
Descripción	Un usuario administrador elimina un usuario.
Actores	Administrador.
Precondiciones	El usuario administrador debe tener una sesión abierta en el sistema y el usuario que quiere eliminar debe existir en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono con forma de llave inglesa y este le lleva a la página de administración. Este botón solamente es visible por los usuarios administradores. 2.- El usuario hace click en la pestaña ”Usuarios” y se le despliega un buscador en el que se puede filtrar por nombre completo y por nombre de usuario. 3.- El usuario hace click en el botón ”eliminar” del usuario que quiere borrar. Se le muestra al usuario una ventana de confirmación de la acción. 4.- El usuario acepta el mensaje de confirmación. 5.- Se cierra la ventana modal, el usuario se elimina de la base de datos y desaparece de la lista de usuarios.
Flujo alternativo	1.- El usuario administrador cierra o cancela la ventana de confirmación. 2.- Se cierra la ventana modal, el usuario no se elimina de la base de datos y no desaparece de la lista de usuarios.
Postcondiciones	El usuario se elimina del sistema.

Cuadro 6.7: CU-107: Eliminar usuario

CU-108: Seguir a un usuario	
Descripción	Un usuario se convierte en seguidor de otro usuario.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	1.- Los dos usuarios, seguidor y seguido, deben estar registrados en el sistema. 2.- El usuario seguidor debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario seguidor accede al perfil del usuario seguido desde cualquier punto de la de la aplicación. 2.- El usuario hace click en el botón ”Seguir” del perfil del usuario seguido. 3.- Se registra en base de datos la nueva relación entre los dos usuarios, seguidor y seguido. El texto del botón cambia a ”Dejar de seguir”.
Flujo alternativo	1.- El usuario seguidor al momento de entrar al perfil del usuario ya lo está siguiendo por lo que se le muestra el botón ”Dejar de seguir”.
Postcondiciones	En el sistema queda registrada la nueva relación entre los dos usuarios.

Cuadro 6.8: CU-108: Seguir a un usuario

CU-109: Dejar de seguir a un usuario	
Descripción	Un usuario deja de ser seguidor de otro usuario.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	1.- Los dos usuarios, seguidor y seguido, deben estar registrados en el sistema. 2.- El usuario seguidor debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario seguidor accede al perfil del usuario seguido desde cualquier punto de la de la aplicación. 2.- El usuario hace click en el botón ”Dejar de seguir” del perfil del usuario seguido. 3.- Se elimina de la base de datos la relación entre los dos usuarios, seguidor y seguido. El texto del botón cambia a ”Seguir”.
Flujo alternativo	1.- El usuario seguidor al momento de entrar al perfil del usuario todavía no lo está siguiendo por lo que se le muestra el botón ”Seguir”.
Postcondiciones	En el sistema se elimina la relación entre los dos usuarios.

Cuadro 6.9: CU-109: Dejar de seguir a un usuario

CU-110: Cargar actividad reciente de usuarios seguidos	
Descripción	Un usuario entra en la pantalla principal y se carga la actividad reciente de los usuarios a los que sigue.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe estar registrado y debe tener una sesión abierta en el sistema.
Flujo básico	1.- Cuando el usuario entra en la pantalla principal de la aplicación se lanza una petición para recuperar de BD la actividad reciente de los usuarios a los que sigue. 2.- Esta actividad puede ser de varios tipos: a.- Un usuario seguido ha puntuado una película. b.- Un usuario seguido le ha recomendado una película. c.- Un usuario seguido ha empezado a seguir a otro 3.- La lista de las cinco primeras “actividades” se presenta en la pantalla principal, dentro de un marco rectangular. Se presentan de manera distinta cada uno de los tipos en orden cronológico descendente. 4.- En cada representación de cada actividad sale el nombre del usuario al que pertenece la “actividad”, en el que se puede hacer click para disparar el caso de uso CU-104: Ver perfil de usuario. 5.- En cada representación de cada actividad en la que aplique, se muestra la representación de una película en la que se puede hacer click para disparar el caso de uso CU-204: Ver detalle de película. 6.- En la parte inferior del marco rectangular se presenta un botón con el texto ”Ver más”, en el que se puede hacer click para disparar el caso de uso CU-111: Mostrar pantalla actividades recientes.
Flujo alternativo	1.- El usuario no tiene ninguna actividad reciente relacionada debido bien a que no sigue a ningún otro usuario, o bien porque los usuarios a los que sigue todavía no han realizado ninguna de las acciones definidas. 2.- Se le muestra al usuario el recuadro vacío.
Postcondiciones	El usuario puede consultar la actividad reciente de los usuarios a los que sigue.

Cuadro 6.10: CU-110: Cargar actividad reciente de usuarios seguidos

CU-111: Mostrar pantalla actividades recientes	
Descripción	Un usuario entra en la pantalla de actividades recientes y se carga la actividad reciente de todos los usuarios a los que sigue.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario accede a la pantalla de actividades recientes pulsando el botón ”Ver más” del recuadro de actividades recientes de la pantalla principal. 2.- Se lanza una petición para recuperar de base de datos todas las actividades realizadas por los usuarios que sigue. 3.- Estas actividades pueden ser de varios tipos: a.- Un usuario seguido ha puntuado una película. b.- Un usuario seguido le ha recomendado una película. c.- Un usuario seguido ha empezado a seguir a otro 4.- Se le presentan las actividades recuperadas al usuario divididas en tres pestañas que se corresponden con cada uno de los tres tipos de actividad. 5.- En cada representación de cada actividad sale el nombre del usuario al que pertenece la “actividad”, en el que se puede hacer click para disparar el caso de uso CU-104: Ver perfil de usuario. 6.- En cada representación de cada actividad en la que aplique, se muestra la representación de una película en la que se puede hacer click para disparar el caso de uso CU-204: Ver detalle de película.
Flujo alternativo	1.- El usuario no tiene ninguna actividad relacionada debido bien a que no sigue a ningún otro usuario, o bien porque los usuarios a los que sigue todavía no han realizado ninguna de las acciones definidas. 2.- Se le muestran al usuarios las tres pestañas vacías.
Postcondiciones	El usuario puede consultar todas las actividades recientes relacionadas separadas por su tipo.

Cuadro 6.11: CU-111: Mostrar pantalla actividades recientes

6.3.2 Casos de uso de películas

CU-201: Crear película	
Descripción	Un usuario administrador crea una nueva película.
Actores	Administrador.
Precondiciones	El usuario administrador debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono con forma de llave inglesa y este le lleva a la página de administración. Este botón solamente es visible por los usuarios administradores. 2.- El usuario hace click en la pestaña ”Películas” y se le despliega un buscador en el que se puede filtrar todas las películas del sistema por título. 3.- El usuario hace click en el botón ”Agregar película”. Se le abre una ventana modal con todos los datos de película necesarios. 4.- El usuario rellena los campos correctamente y pulsa el botón ”Guardar”. El sistema valida todos los campos introducidos por el usuario y los almacena en la base de datos. 5.- Se cierra la ventana modal.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario administrador introduce un dato no válido o en blanco. 2.- Se informa al usuario administrador de qué datos son incorrectos y cómo debe proceder para solucionarlo. Flujo alternativo 2: 1.- El usuario administrador pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se almacenan los datos introducidos.
Postcondiciones	La película queda registrada en el sistema.

Cuadro 6.12: CU-201: Crear película

CU-202: Modificar película	
Descripción	Un usuario administrador edita los datos de una película.
Actores	Administrador.
Precondiciones	El usuario administrador debe tener una sesión abierta en el sistema y la película que quiere modificar debe existir en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono con forma de llave inglesa y este le lleva a la página de administración. Este botón solamente es visible por los usuarios administradores. 2.- El usuario hace click en la pestaña ”Películas” y se le despliega un buscador en el que se puede filtrar todas las películas del sistema por título. 3.- El usuario hace click en el botón ”editar” de la película que quiere modificar. Se le abre una ventana modal igual a la que añade una película pero con los datos cubiertos. 4.- El usuario modifica los campos que desea correctamente y pulsa el botón ”Guardar”. El sistema valida todos los campos introducidos por el usuario y los actualiza en la base de datos. 5.- Se cierra la ventana modal.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario administrador introduce un dato no válido o en blanco. 2.- Se informa al usuario administrador de qué datos son incorrectos y cómo debe proceder para solucionarlo. Flujo alternativo 2: 1.- El usuario administrador pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se modifican los datos de la película seleccionada.
Postcondiciones	Los cambios realizados sobre la película quedan almacenados en el sistema.

Cuadro 6.13: CU-202: Modificar película

CU-203: Eliminar película	
Descripción	Un usuario administrador elimina una película.
Actores	Administrador.
Precondiciones	El usuario administrador debe tener una sesión abierta en el sistema y la película que quiere eliminar debe existir en el sistema.
Flujo básico	1.- El usuario hace click en el botón con el icono con forma de llave inglesa y este le lleva a la página de administración. Este botón solamente es visible por los usuarios administradores. 2.- El usuario hace click en la pestaña ”Películas” y se le despliega un buscador en el que se puede filtrar todas las películas del sistema por título. 3.- El usuario hace click en el botón ”eliminar” de la película que quiere borrar. Se le muestra al usuario una ventana de confirmación de la acción. 4.- El usuario acepta el mensaje de confirmación. 5.- Se cierra la ventana modal, la película se elimina de la base de datos y desaparec de la lista de películas.
Flujo alternativo	1.- El usuario administrador cierra o cancela la ventana de confirmación. 2.- Se cierra la ventana modal, la película no se elimina de la base de datos y no desaparece de la lista de películas.
Postcondiciones	La película se elimina del sistema.

Cuadro 6.14: CU-203: Eliminar película

CU-204: Ver detalle de película	
Descripción	Un usuario consulta el detalle de una película.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe tener una sesión abierta en el sistema y la película que quiere consultar debe existir en el sistema.
Flujo básico	1.- El usuario hace click en el título de una película desde cualquier punto de la aplicación. 2.- Se recupera la información de la película de base de datos y se le muestra al usuario. La información que se le muestra al usuario está compuesta por: a.- Información propia de la película (título, poster, año de estreno, dirección, reparto, etc..) b.- Puntuación media de la película en el sistema. c.- Puntuación media de la película en otras páginas conocidas. d.- Críticas hechas por otros usuarios.
Flujo alternativo	
Postcondiciones	El usuario pude ver la información detallada de la película.

Cuadro 6.15: CU-204: Ver detalle de película

CU-205: Puntuar una película	
Descripción	Un usuario le da una puntuación del 1 al 10 a una película según su criterio.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe tener una sesión abierta en el sistema y la película que quiere puntuar debe existir en el sistema.
Flujo básico	1.- El usuario accede al detalle de una película. 2.- El usuario pasa por encima con el ratón por el desplegable con el texto ”Dar puntuación” situado debajo de la puntuación media de la película. 3.- El usuario selecciona el número de estrellas que le quiere dar a la película de entre las opciones desplegadas. Las opciones desplegadas son de 1 a 10 estrellas. 4.- Se envía y se almacena la información de la puntuación dada por el usuario a la película. 5.- Se actualiza la vista para mostrar la puntuación dada por el usuario y la fecha.
Flujo alternativo	1.- Si el usuario ya ha puntuado con anterioridad la película no se le muestra el texto ”Dar puntuación” si no que se le muestra la puntuación dada y la fecha. 2.- Si el usuario desea modificar su puntuación puede continuar con el caso de uso desde el punto 3 del flujo básico.
Postcondiciones	La puntuación que ha dado un usuario a una película se almacena en base de datos.

Cuadro 6.16: CU-205: Puntuar una película

CU-206: Recomendar una película	
Descripción	Un usuario le recomienda una película a otro usuario al que sigue.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe tener una sesión abierta en el sistema y la película que quiere recomendar debe existir en el sistema.
Flujo básico	1.- El usuario accede al detalle de una película. 2.- El usuario hace click en el botón con el texto ”Recomendar” situado en la parte superior derecha de la pantalla de detalle. 3.- Se le abre una ventana modal en la que puede seleccionar el usuario o los usuarios a los que quiere recomendarle la película de entre todos a los que sigue. 4.- El usuario pulsa el botón ”Aceptar” de la ventana modal. 5.- Se envía la información al servidor y se almacena en base de datos. 6.- Se cierra la ventana modal. 7.- El usuario o usuarios a los que ha recomendado esta película podrán ver en su pantalla principal, en la sección de ”Actividades recientes”, que el usuario les ha recomendado la película.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario no tiene otros usuario seguidos. 2.- En la ventana modal no se le muestra ningún usuario que pueda seleccionar. 3.- Realice la acción que realice el usuario sobre esta ventana modal no se produce la recomendación. Flujo alternativo 2: 1.- El usuario pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se produce la recomendación.
Postcondiciones	Los usuarios seleccionados puede ver que el usuario les ha recomendado la película.

Cuadro 6.17: CU-206: Recomendar una película

CU-207: Crear crítica	
Descripción	Un usuario crítico realiza una crítica de una película.
Actores	Administrador y crítico.
Precondiciones	El usuario debe tener una sesión abierta en el sistema y la película a la que le va a realizar la crítica debe estar registrada en el sistema.
Flujo básico	1.- El usuario entra en el detalle de alguna película. 2.- El usuario hace click en el botón ”Añadir review” del detalle de la película. 3.- Se le abre una ventana modal al usuario donde puede indicar por un lado la puntuación que le da a la película y también puede desarrollar la crítica propiamente dicha en un cuadro de texto. 4.- El usuario cubre ambos campos y pulsa en el botón ”Guardar”. 5.- Se envía y almacena la información de la crítica y se cierra la ventana modal.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se crea la crítica de la película. Flujo alternativo 2: 1.- El usuario deja en blanco o con valor inválido alguno de los dos campos. 2.- Se le muestra un mensaje al usuario indicando como proceder para solucionarlo y se le impide el guardado de la crítica.
Postcondiciones	El usuario crea una crítica de una película.

Cuadro 6.18: CU-207: Crear crítica

CU-207: Editar crítica	
Descripción	Un usuario crítico realiza una modificación sobre una crítica de una película hecha por él mismo anteriormente.
Actores	Administrador y crítico.
Precondiciones	El usuario debe tener una sesión abierta en el sistema. La crítica que va a editar debe existir en el sistema.
Flujo básico	1.- El usuario entra en el detalle de alguna película. 2.- El usuario hace click en el botón ”Editar review” del detalle de la película. 3.- Se le abre una ventana modal al usuario donde se le pre-cargan los campos puntuación y texto de la crítica con los datos proporcionados anteriormente por el propio usuario. 4.- El usuario modifica los campos y pulsa en el botón ”Guardar”. 5.- Se envía y almacena la nueva información de la crítica y se cierra la ventana modal.
Flujo alternativo	Flujo alternativo 1: 1.- El usuario pulsa en el botón ”Cancelar” o en la aspa para cerrar la ventana modal 2.- Se cierra la ventana modal y no se modifica la crítica de la película. Flujo alternativo 2: 1.- El usuario deja en blanco o con valor inválido alguno de los dos campos. 2.- Se le muestra un mensaje al usuario indicando como proceder para solucionarlo y se le impide el guardado de la crítica.
Postcondiciones	El usuario modifica una crítica de una película.

Cuadro 6.19: CU-207: Editar crítica

CU-208: Cargar películas mejor valoradas	
Descripción	Un usuario entra en la pantalla principal y se cargan las películas mejor valoradas por los usuarios.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe estar registrado y debe tener una sesión abierta en el sistema.
Flujo básico	1.- Cuando el usuario entra en la pantalla principal de la aplicación se lanza una petición para recuperar las películas mejor valoradas. 2.- Se recupera la información de las 5 primeras películas ordenadas de manera descendente por la puntuación media calculada a partir de todas las puntuaciones dadas por los usuarios. La información que se recupera de estas películas es: su título, director/es, su poster y su puntuación media. 3.- Esta información se le presenta al usuario en la pantalla principal dentro de un recuadro. 4.- El usuario puede hacer click en el título de cualquiera de las 5 películas lo que dispara el caso de uso CU-204: Ver detalle de película. 5.- En la parte inferior del marco rectangular se presenta un botón con el texto ”Ver más”, en el que se puede hacer click para disparar el caso de uso CU-209: Mostrar pantalla mejor valoradas
Flujo alternativo	–
Postcondiciones	El usuario puede consultar las películas mejor valoradas.

Cuadro 6.20: CU-208: Cargar películas mejor valoradas

CU-209: Mostrar pantalla mejor valoradas	
Descripción	Un usuario entra en la pantalla de películas mejor valoradas.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe estar registrado y debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario accede a la pantalla de películas mejor valoradas pulsando el botón ”Ver más” del recuadro de películas mejor valoradas de la pantalla principal 2.- Se lanza una petición de base de datos para recuperar todas las películas ordenadas de forma descendente por la puntuación media de la película. Estas se le muestran al usuario de forma paginada. 3.- De cada película se muestra su título, director/es, número de veces votada (vista) y su puntuación media. 4.- En la parte superior de la pantalla se muestran varios filtros: a.- Año de estreno de la película b.- Título c.- Género 5.- El usuario puede utilizar estos filtros para realizar búsquedas más específicas mediante el CU-210: Buscar películas en la pantalla mejor valoradas. 6.- El usuario puede hacer click en el título de cualquiera de las 5 películas lo que dispara el caso de uso CU-204: Ver detalle de película.
Flujo alternativo	–
Postcondiciones	El usuario puede consultar todas las películas ordenadas de mejor a peor valoradas.

Cuadro 6.21: CU-209: Mostrar pantalla mejor valoradas

CU-210: Buscar películas en la pantalla mejor valoradas	
Descripción	Un usuario entra en la pantalla de películas mejor valoradas y realiza una búsqueda.
Actores	Usuario registrado, crítico y administrador.
Precondiciones	El usuario debe estar registrado y debe tener una sesión abierta en el sistema.
Flujo básico	1.- El usuario accede a la pantalla de películas mejor valoradas pulsando el botón ”Ver más” del recuadro de películas mejor valoradas de la pantalla principal 2.- En la parte superior de la pantalla se muestran varios filtros: a.- Año de estreno de la película b.- Título c.- Género 3.- El usuario cubre alguno o varios de los campos de filtro y pulsa el botón ”Filtrar”. 4.- Se realiza una petición a la base de datos que recupera las películas que cumpla todos los criterios introducidos por el usuario en los filtros disponibles. 5.- Se le presentan al usuario estas películas ordenadas de forma descendente por la puntuación media y paginadas.
Flujo alternativo	1.- Los criterios introducidos por el usuario no se corresponden con ninguna película almacenada en el sistema. 2.- Se le muestra una lista vacía al usuario.
Postcondiciones	El usuario puede consultar y filtrar las películas ordenadas de mejor a peor valoradas.

Cuadro 6.22: CU-210: Buscar películas en la pantalla mejor valoradas

Los casos de uso **CU-211: Cargar películas más vistas**, **CU-212: Mostrar pantalla más vistas** y **CU-213: Buscar películas en la pantalla más vistas** al ser prácticamente los mismos casos de uso que CU-208, CU-209 y CU-210 respectivamente excepto que el criterio de ordenación cambia de las mayores puntuaciones medias al mayor número de puntuaciones. Cambia también el dato que se recupera, que pasa a ser el número veces que se ha puntuado. Por este motivo y para evitar que se haga muy extenso se ha prescindido de escribirlos en forma de cuadro.

Capítulo 7

Diseño de la aplicación

En este capítulo se detallarán los aspectos y decisiones más importantes llevadas a cabo en relación al diseño de la aplicación y su arquitectura.

Para el desarrollo de los requisitos expuestos en el capítulo anterior se han utilizado diferentes patrones y buenas prácticas. Destaca el uso del patrón *Model-View-Controller* o Modelo-Vista-Controlador ya que marca el diseño de toda la aplicación.

7.1 Patrones de diseño

7.1.1 Model-View-Controller

Model-View-Controller (MVC) [26] es un patrón arquitectónico usado generalmente para el desarrollo de aplicaciones web. Divide el código en tres capas principales: *model* o modelo, *view* o vista y *controller* o controlador.

Originalmente, el patrón de diseño MVC fue introducido con el entorno de programación Smalltalk como una forma de estructurar aplicaciones interactivas de manera modular en 1988. Pero ha ido evolucionando con el tiempo y el patrón ya no se entiende exactamente igual a cuando fue formulado. A continuación se explicará brevemente como se interpreta el patrón MVC dentro de los *frameworks* de aplicaciones web.

El componente **modelo**, en general, se encarga de mantener el estado de la aplicación. Responde a peticiones que le llegan desde el controlador realizando diversas acciones. Entre ellas destaca la realización de operaciones sobre una base de datos para mantenerla, ejecución de lógica de la aplicación y el menjo de la interacción con sistema externos como puede ser un servicio web.

El componente **vista** se encarga de presentar una interfaz de usuario. Le presenta la información al usuario y realiza las peticiones necesarias para recuperar esta información del modelo. Proporcionar una experiencia interactiva del lado del cliente para el usuario. Le permite interactuar con la aplicación a través de formularios y distintos controles.

El componente **controlador** actúa como frontal del modelo de la aplicación reciviendo las solicitudes entrantes y enrutándolas al manejador adecuado. Se encarga de la validación de los parámetros de las solicitudes recibidas y también de orquestar el manejo de solicitudes invocando elementos del modelo apropiados.

La principal ventaja del uso de este patrón es el desacople de los componentes principales permitiendo así la reutilización de código. Por ejemplo, se puede reutilizar la misma parte modelo y controlador para varias vistas distintas. También facilita el trabajo en paralelo, un desarrollador puede estar trabajando en la vista y el otro en el modelo.

7.1.2 Facade

El patrón de diseño *Facade* [27] es un tipo de patrón estructural. Proprociona a un cliente de un subsitema complejo una interfaz de alto nivel simplificada para minimizar las dependencias que pueda tener dicho cliente con varias partes o clases del subsitema.

7.1.3 Repository

El patrón *Repository* [28] define una capa intermedia que separa la lógica de negocio de las interacciones con la fuente de datos subyacente. Esta fuente de datos puede ser una base de datos, un ficheros, etc…

Esta capa se encarga de recuperar los datos del origen, mapearlos a las entidades de negocio y persistir cambios que se hagan en las entidades. Esto permite centralizar toda la lógica de datos en un punto evitando así posibles errores. Además, hace que la aplicación sea más flexible a los cambios que pueda experimentar con el paso del tiempo. Si se cambia la fuente de datos, habría que modificar esta capa pero se podríaa reutilizar toda la lógica de negocio.

7.1.4 Data Transfer Object (DTO)

Un *Data Transfer Object* [29] es un objeto que permite pasarle a una función un conjunto de datos agrupados. Los DTOs son meros contenedores de datos, no deben contener nigún tipo de lógica de negocio, únicamente validaciones sencillas para mantener la coherencia in-

terna.

Utilizar DTOs es especialmente útil cuando la función llamada es una función que se ejecuta en un sistema externo remoto, ya que permite aumentar el renidimiento al realizar una única llamada con todos los datos necesarios agrupados en lugar de varias. En la mayoría de los escenarios, una llamada remota que lleva una mayor cantidad de datos lleva prácticamente el mismo tiempo que una llamada que solo lleva una pequeña cantidad de datos.

7.1.5 Inversion of Control (IOC)

Inversion of Control (IOC) [30] o inversión de control es un patrón de diseño cuyo principio dicta que las dependencias explícitas de los componentes deben eliminarse del código.

En muchos entornos, los componentes son los encargados de localizar los otros componentes de los que dependen para su ejecución. Esto se conoce como el patrón de localización de servicios. Utilizando IOC, en lugar de ser el componente el que llama a una infraestructura en tiempo de ejecución para vincularse a otros componentes, la infraestructura es la que llama al componente y le proporciona los recursos necesarios para se ejecute.

Este patrón ha sido implementado en este proyecto utilizando el entorno de Spring.

7.1.6 Page-by-Page Iterator

El patrón *Page-by-Page Iterator* [31] permite al usuario recuperar una lista de datos especificando el número máxmimo de elementos que espera recibir.

Lo que se busca al emplear este patrón es aumentar la eficiencia de una aplicación que quiera trasmitir largas listas de datos. La transferencia de listas grandes significa más tiempo, lo que resulta en una mala experiencia para el usuario. A la hora de decidir si utilizar o no este patrón es importante tener en cuenta si el usuario está realmente interesado en consultar la lista completa de datos. Además, el cliente que solicita la lista puede tener poca cantidad de memoria o una pantalla pequeña, por lo tanto puede ser que el cliente tampoco sea capaz de manejar listas tan grandes.

7.2 Arquitectura del sistema

Como se ha adelantado en la sección 4.1 la arquitectura global del sistema sigue el patrón de arquitectura MVC. Esta se divide en tres partes diferenciadas, cada una de las cuales agrupa un conjunto de funcionalidades relacionadas entre si para que la aplicación sea modulable

y permitir así que sea mantenible. En la figura 4.1 presentada con anterioridad, se pueden observar a grandes rasgos las funcionalidades que agrupa cada uno de los tres módulos, que a continuación se explicarán en profundidad.

7.2.1 Modelo

La capa modelo es la parte de la aplicación que encapsula toda la lógica de negocio y los accesos a base de datos. En esta capa de la arquitectura es donde se almacenan, modifican y eliminan todos los datos que maneja la aplicación.

Entidades persistentes

En la figura 7.1, se puede ver el conjunto de todas las entidades persistentes de la aplicación representadas utilizando el lenguaje de modelado UML.

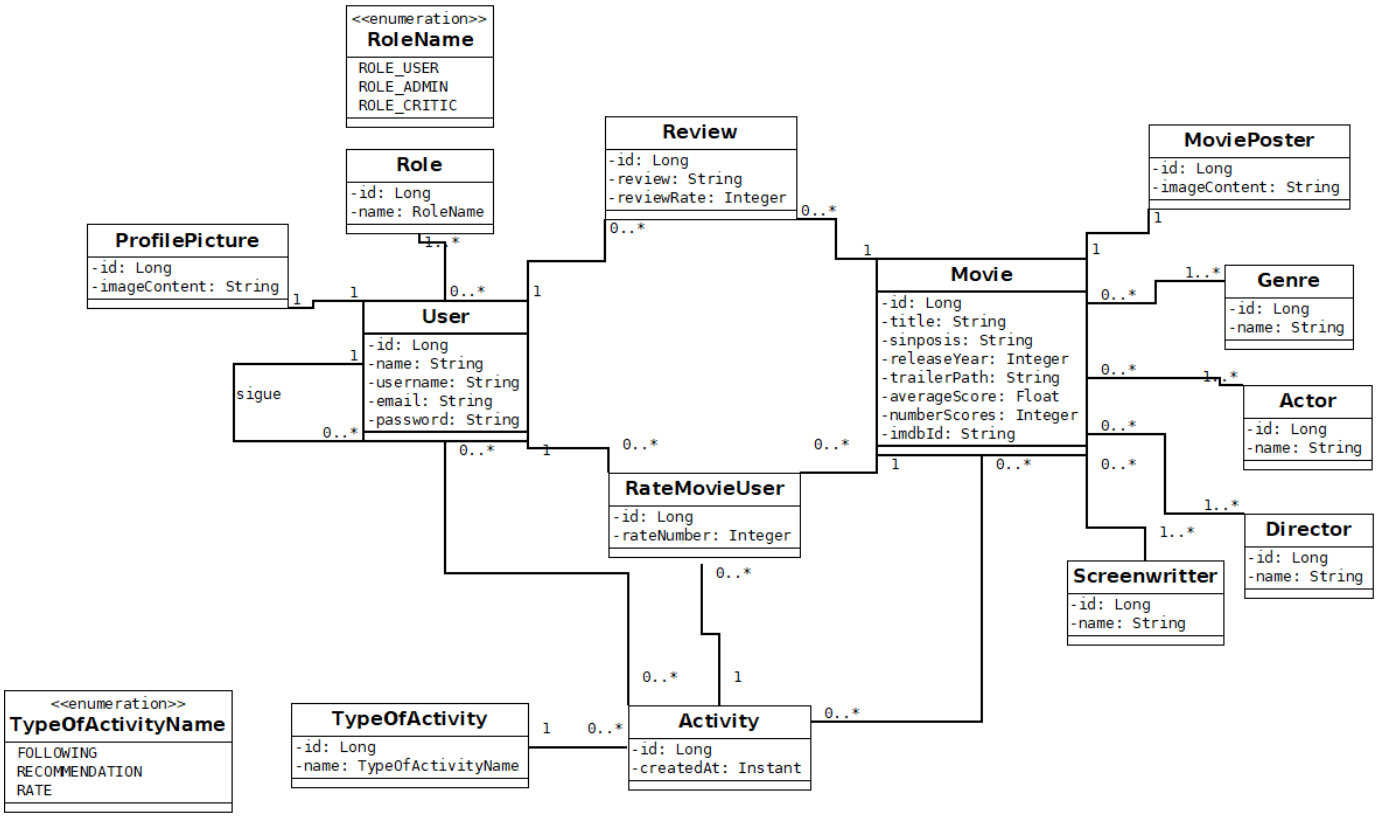


Figura 7.1: Diagrama de entidades persistentes

A continuación se detallarán brevemente cada una de las entidadades:

- **User**: Representa los datos de un usuario existente en el sistema.
- **Role**: Representa un rol que puede tener un usuario en la aplicación.
- **RoleName**: Es un tipo enumerado que representa los tres tipos de roles que existen en la aplicación.
- **ProfilePicture**: Representa la foto de perfil de un usuario del sistema.
- **Movie**: Representa una película existente en el sistema.
- **MoviePoster**: Representa el poster de una película existente en el sistema.
- **Genre**: Representa un género (comedia, drama, etc...) que puede estar asociado a cualquier película del sistema.
- **Actor**: Representa los datos de un actor existente en el sistema. Puede estar o no asociado a alguna película
- **Director**: Representa los datos de un director existente en el sistema. Puede estar o no asociado a alguna película
- **Screenwritter**: Representa los datos de un guionista existente en el sistema. Puede estar o no asociado a alguna película
- **Review**: Representa una reseña que la hace un usuario a una película. Consta de un número que representa la puntuación de la película y un texto explicativo.
- **RateMovieUser**: Representa la puntuación que le da un usuario a una película.
- **Activity**: Representa una acción realizada por un usuario dentro de la aplicación. Puede estar relacionada con una película o con otro usuario.
- **TypeOfActivity**: Representa los tipos de actividades que existen en el sistema.
- **TypeOfActivityName**: Es un tipo enumerado que representa los tres tipos de acitividades que existen en la aplicación.

Capa de acceso a datos

Una vez vistas las entidades persistentes que forman la aplicación, a continuación, se expondrá el diseño que ha realizado para la capa de acceso a datos. En esta capa se ha hecho uso del patrón *Repository*, explicado en la sección 7.1.3.

En la figura 7.2, se pueden ver todos los repositorios de la aplicación a excepción del *MovieRepository*, ya que este es un caso especial que se explica más adelante. Todos ellos extienden la clase genérica JpaRepository<T, ID> de *Spring*. Donde **T** es la entidad persistente, mientras que el parámetro **ID** define el tipo de dato que identifica a la entidad persistente. Al extender de esta clase, ya se puede realizar operaciones básicas CRUD para añadir, modificar y eliminar entidades.

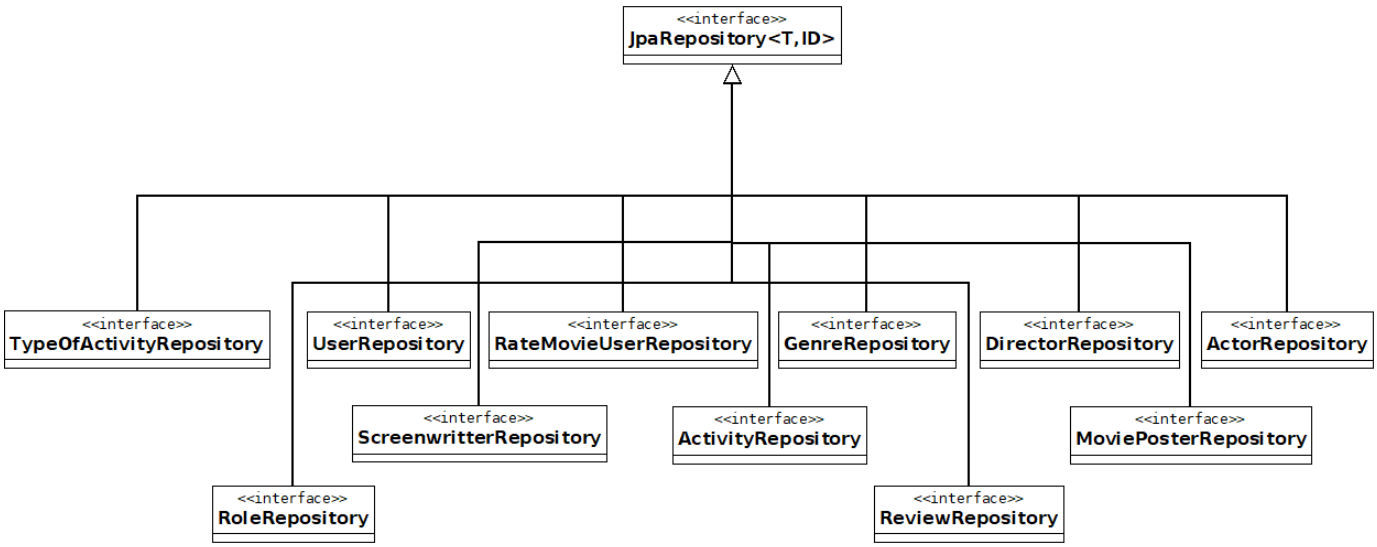


Figura 7.2: Diagrama de repositorios

Como se puede ver en la figura anterior, todos los *repositories* son interfaces. Al utilizar los *JpaRepository* de Spring Data es posible crear los llamados *Query methods* o ”métodos Query” gracias a los Query DSL, acrónimo de *Domain Specific Language*. Estos permiten la creación de métodos de interfaz Java utilizando en sus nombres palabras clave junto con atributos de las entidades JPA para que los Query DSL realicen el trabajo de la implementación de la consulta, quitando mucho trabajo de condificación al desarrollador.

Otra forma de implementar las funciones de la capa de acceso a datos ha sido utilizando la anotación *@Query* de Spring Data JPA. Utilizando esta anotación, se especifica explicitamente en los métodos de la interfaz la consulta en lenguaje JPQL o SQL que será ejecutada cuando se llame a dichos métodos.

En la siguientes figuras se muestra el detalle de las interfaces de los *repositories* introducidos en la figura 7.2.

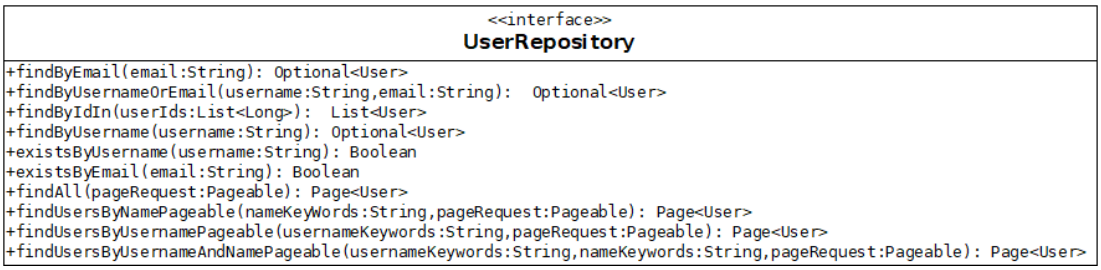


Figura 7.3: UserRepository

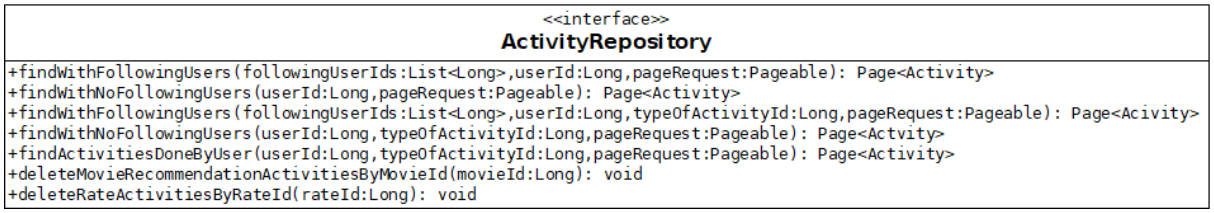


Figura 7.4: ActivityRepository

Figura 7.5: TypeOfActivityRepository

Figura 7.6: RoleRepository

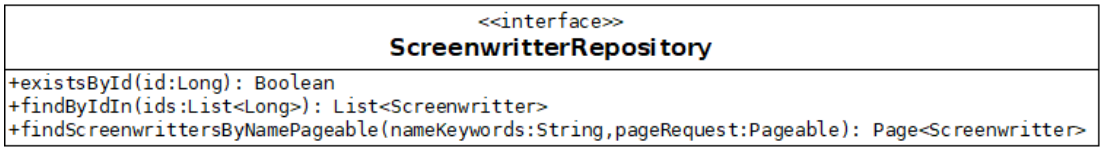


Figura 7.7: ScreenwritterRepository

Figura 7.8: RateMovieUserRepository

Figura 7.9: GenreRepository

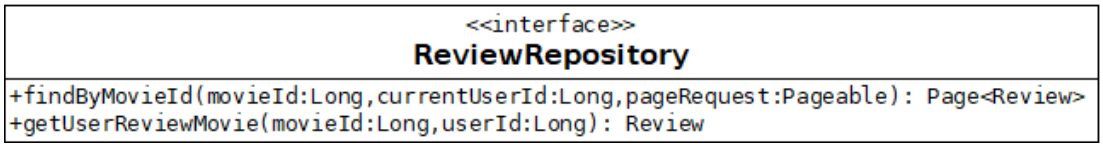


Figura 7.10: ReviewRepository

Figura 7.11: DirectorRepository

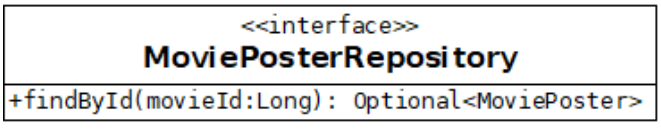


Figura 7.12: MoviePosterRepository

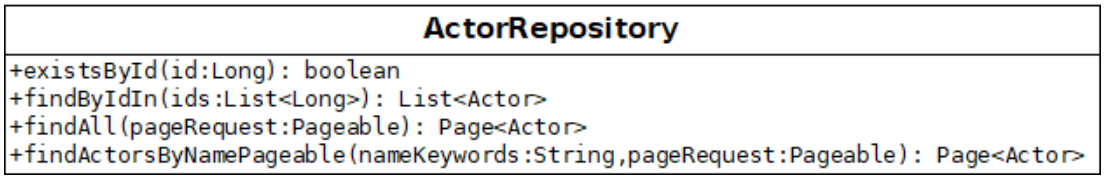


Figura 7.13: ActorRepository

En cuanto al *MovieRepository* se ha seguido un enfoque distinto. Debido a la complejidad que requieren las búsquedas de películas utilizando varios filtros, se ha realizado una implementación personalizada del método propio *findMovieByCriteriaPaginated*, definido en la interfaz *MovieRepositoryCustom*. Como en los otros casos también se han utilizado *Query methods* y la anotación *@Query* en la interfaz *MovieRepository* para casos más sencillos. En la siguiente figura se puede ver el diseño que se ha utilizado.

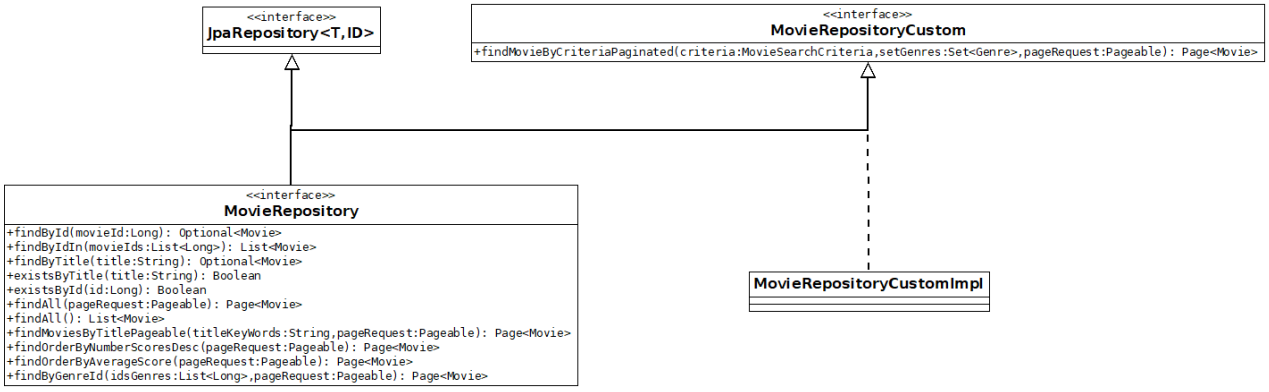


Figura 7.14: MovieRepository

Capa de servicio

Los *repositories* vistos con anterioridad serán usados para la capa de abstracción inmediatamente superior, la capa de servicio. Esta capa se encarga de implementar la lógica de negocio y exponerla en forma de funciones para que pueda ser llamada por las capas superiores. En esta aplicación se han dividido los requisitos en 9 servicios distintos. Se han dividido de forma que las operaciones que le afecten a una misma entidad del dominio estarán en el mismo servicio. A continuación se pasará a dar una breve explicación de cada uno de ellos.

Primero se introduce el servicio más importante, **MovieService**. En la figura 7.15 se puede ver la fachada de este, con todos los métodos que implementa. Utilizando este servicio se puede crear películas, actualizarlas, eliminarlas, recuperarlas, realizar búsquedas paginadas, realizar distintas comprobaciones sobre ellas y puntuarlas. Además, en este servicio se han incluido varias operaciones sobre los géneros de las películas, como puede ser recuperarlos por su identificador o comprobar si existen en el sistema.

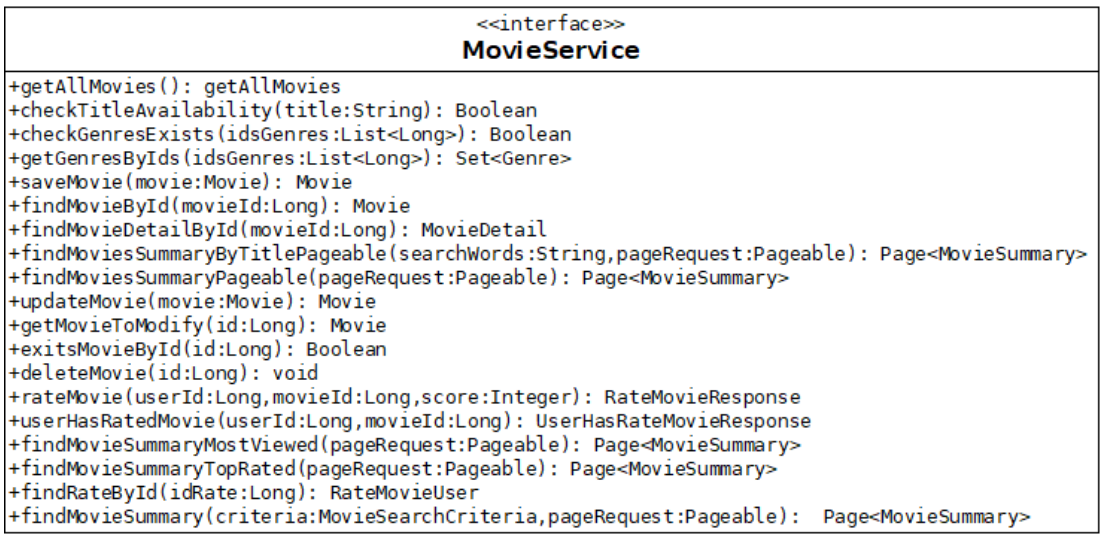


Figura 7.15: MovieService

En la figura 7.16 se puede ver la fachada del servicio **UserService**. Este se encarga de realizar todas las operaciones relacionas con usuarios. Además de poder crear, modificar y eliminar usuarios, se han creado operaciones para realizar búsquedas sobre ellos. Se han creado operaciones también para el manejo de los roles de usuario. Además, es posible conocer si un usuario siga a otro, hacer que empiece a seguirlo o deje de hacerlo.

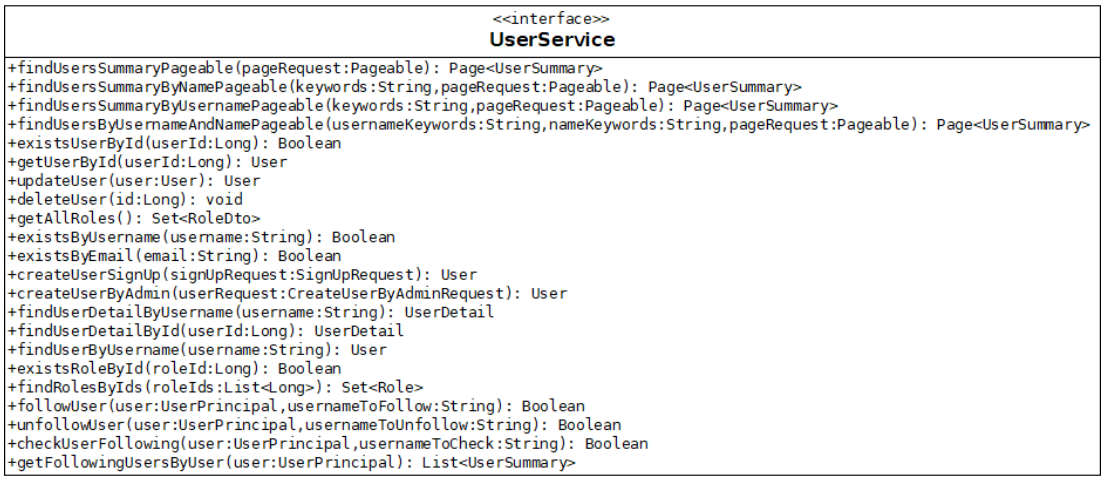


Figura 7.16: UserService

A continuación se presenta el **ActivityService** cuya fachada se puede ver en la figura 7.17. Este servicio se utiliza para crear actividades de los tres tipos que se han definido en la aplicación: un usuario sigue a otro, un usuario recomienda una película y un usuario puntúa una película. También se han definido dos operaciones de búsqueda, una para recuperar las actividades para mostrar en el *feed* de un usuario las actividades realizadas por usuarios a los que sigue. Y otra para encontrar las actividades que ha realizado un usuario para mostrarlas en la ventana de su perfil. Ha sido necesario definir también una operación para eliminar toda la actividad relacionada con una película, necesaria para el caso de uso CU-203: Eliminar película.

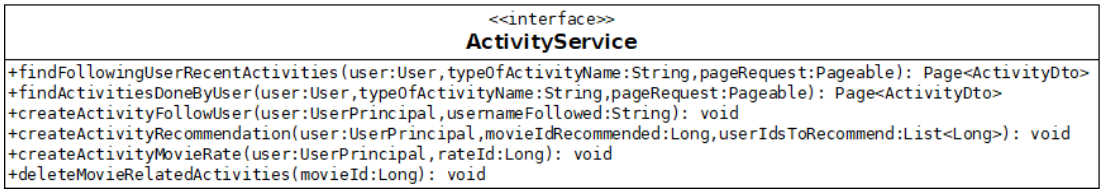


Figura 7.17: ActivityService

Los servicios **ActorService, DirectorService y ScreenwritterService** define operaciones prácticamente idénticas para el manejo de las entidades Actor, Director y Screenwritter. En la figura 7.18 se puede ver que se han definido métodos de consulta y un método para comprobar si existe o no la entidad en el sistema a partir de su identificador.

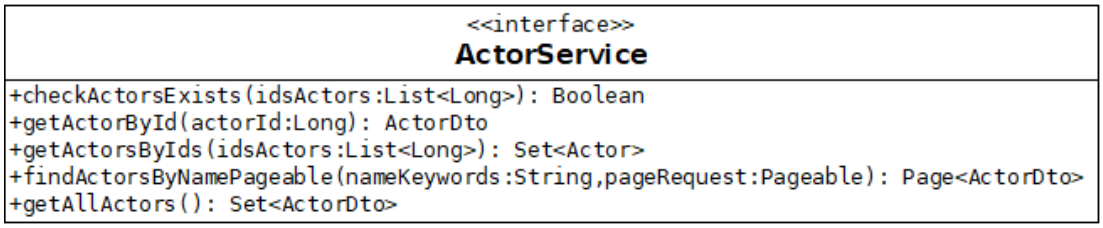


Figura 7.18: ActorService

Se ha decidido crear un servicio para recuperar y crear *reviews* que los usuarios le pueden hacer a las películas. Este se llama **ReviewService**, y en la figura 7.19 se puede ver su fachada.

Figura 7.19: ReviewService

Se ha creado un servicio denominado **GenreService**, con una operación para recuperar todos los géneros de películas disponibles en el sistema, y otra para recuperar estos géneros dado un identificador. Se ha creado otro más llamado **MoviePosterService** con un método para guardar una imágen de poster de una película y otro para recuperar esta imágen a partir de su identificador. Sus fachas se pueden ver en las figuras 7.20 y 7.21 respectivamente.

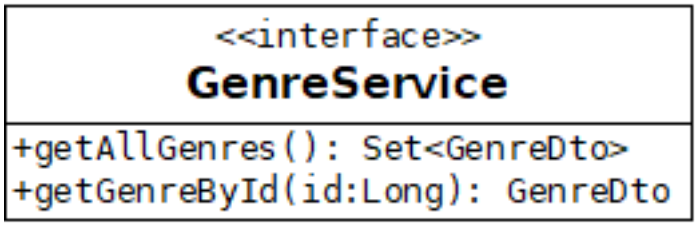


Figura 7.20: GenreService

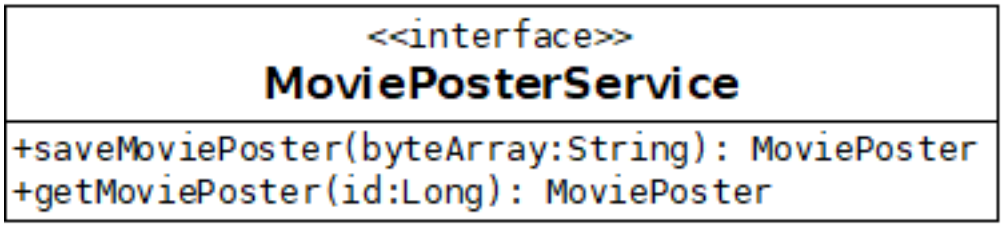


Figura 7.21: MoviePosterService

En la figura 7.22 se puede ver el **OmdbApiService** con una única operación que permite recuperar las puntuaciones de varias webs externas para una película teniendo su identificador.

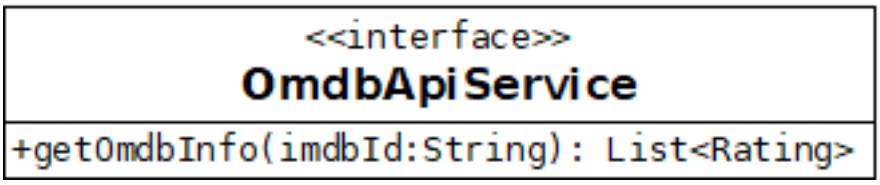


Figura 7.22: OmdbApiService

7.2.2 Controlador

Esta parte de la arquitectura de la aplicación es la encargada de actuar como intermediario entre la parte modelo que encierra la lógica de negocio, y la parte vista, que es la encargada de presentarle la información al usuario.

Como controlador se ha decidido crear un servicio REST con diferentes *endpoints* para separar las distintas funcionalidades. Para el desarrollo de esta parte de la arquitectura del sistema se ha hecho uso de *Spring Framework*. Este se encarga de recibir peticiciones HTTP de los usuarios, decidir qué clase y qué método Java debe manejar la petición y por último responder de forma adecuada según la salida de dicho método. Se han definido 4 clases que actúan como controladores dentro de la aplicación. A continuación se explicaran brevemente:

- **AdminController**. Se ha creado este controlador para poder tener todas juntas las funciones que se encargan de realiar acciones que únicamente pueden hacer los usuarios administradores. Estas acciones entre otras son: crear, modificar y eliminar un usuario; crear, modificar y eliminar pelícuas, etc...
- **AuthController**. Este controlador se encarga únicamente de dos operaciones, autenticar un usuario o registrarlo en el sistema. Se ha creado separado para poder darle acceso a estos dos métodos a usuarios no autenticados.

- **UserController**. Es el encargado de recibir y responder a peticiones de la vista relacionadas con los usuarios. Como puede ser recuperar información del perfil de un usuario, recuperar su *feed*, recuperar los usuarios a los que sigue o seguir/dejar de seguir a otro usuario.
- **MovieController.** Es el controlador que se encarga de recibir y responder a peticiones relacionadas con películas. Estas peticiones pueden ser de recuperación de información de películas o acciones sobre estas, como puntuar una o registrar una crítica de otra.

Dentro de estos controladores se utilizar varias clases DTO tanto para responder a peticione como para recibir parámetros. Esto es posible debido a que *Spring Framework* permite mapear clases Java a formato JSON y viceversa. Se han creado en función de los casos en los que se usan limitando los parámetros de una operación o limitando la cantidad de información con la que se responde a la vista.

De entre todos ellos destacar el uso de un DTO genérico denominado *ApiResponse*. El cual está compuesto simplemente por un *Boolean* indicándo si la operación se ha ejecutado con éxito o no. En caso de que sea no, se tiene también una propiedad *String* que contiene un mensaje descriptivo del error que ha sucedido. Esta clase se utiliza dentro del cuerpo de una respuesta HTTP. La respuesta completa HTTP, constituida por cuerpo, cabeceras y código de estado se construye utilizando la clase ResponseEntity de Spring.

Primero se van a listar las clases que se han utilizado como contenedores de parámetros para realizar llamadas a los métodos definidos en los controladores desde la parte vista.

- **CreateUserByAdminRequest.** Se utiliza para que un usuario administrador cree un usuario.
- **ModifyUserRequest.** Se utiliza para que un usuario administrador modifique un usuario.
- **LoginRequest.** Se utiliza para que un usuario se autentique en el sistema.
- **SignUpRequest.** Se utiliza para que un usuario se registre en el sistema.
- **RegisterMovieRequest.** Se utiliza para que un usuario administrador registre una película.
- **ModifyMovieRequest.** Se utiliza para que un usuario administrador modifique una película existente.
- **RateMovieRequest.** Se utiliza para que un usuario puntúe una película.

- **RegisterReviewRequest.** Se utiliza para que un usuario crítico o administrador cree una crítica a una película.
- **RecommendMovieRequest.** Se utiliza para que un usuario recomiende una película a otros usuarios.

A continuación se explicarán brevemente las clases que se han utilizada para dar respuesta a las peticiones enviadas desde la vista.

- **JwtAuthenticationResponse.** Se utiliza como respuesta a una petición envíada por un usuario para autenticarse en el sistema. Contiene el *token* de autentificación que el usuario almacena y luego utiliza para las siguientes peticiones.
- **RateMovieResponse.** Se utiliza como entidad de respuesta para la operación de puntuar una película. Se le devuelve al usuario la nueva puntuación media de la película, la puntuación que le ha dado, la fecha y el identificador de la puntuación dada por el usuario. Toda esta información es necesaria para el correcto funcionamiento de la parte vista.
- **UserHasRateMovieResponse.** Se utiliza como respuesta ante la petición desde la vista para comprobar si un usuario ya ha puntuado una película previamente.
- **UserReviewMovieResponse.** Se utiliza como respuesta para la petición lanzada desde la vista para comprobar si un usuario ya ha creado una crítica previamente. Si es así, se devuelve dentro de esta clase el objeto ReviewDto que representa la crítica del usuario.
- **CheckUserFollowingResponse.** Se utiliza como respuesta ante la petición desde la vista para comprobar si un usuario sigue a otro.
- **MovieSummary.** Se utiliza como respresentación abreviada de una película para dar respuesta a varias peticiones distintas. Contiene su identificador dentro del sistema, el título, el año de estreno, la lista de directores/as, su puntuación media y el número total de votos.
- **DirectorDto.** Se utiliza como representación de un *Director*. Simplemente contiene su identificador y su nombre.
- **ScreenwritterDto.** Se utiliza como representación de un guionista (*Screenwritter*). Simplemente contiene su identificador y su nombre.
- **ActorDto.** Se utiliza como representación de un *Actor*. Simplemente contiene su identificador y su nombre.

- **GenreDto.** Se utiliza como representación de un género de una película (*Genre*). Simplemente contiene su identificador y su nombre.
- **UserSummary.** Se utiliza como representación abreviada de un usuario para dar respuesta a varias peticiones distintas. Contiene su identificador dentro del sistema, su nombre y nombre de usuario, su foto de perfil y la lista de roles que tiene asignados.
- **RoleDto.** Se utiliza como representación de un rol de un usuario (*Role*). Simplemente contiene su identificador y su nombre.
- **MovieDetail.** Se utiliza como una representación completa de una película. Se utiliza para devolver toda la información de una película necesaria para construir el detalle de una película en la parte vista.
- **ExternalRatingDto.** Se utiliza dentro de la clase *MovieDetail* para representar las puntuaciones que tiene una película en aplicaciones de terceros.
- **ReviewDto.** Representa una crítica de un usuario a una película. Contiene su identificador, el usuario que ha hecho la crítica, el identificador de la película, el texto de la crítica, la puntuación dada en la crítica y la fecha de creación.
- **UserIdentityAvailability.** Se utiliza para responder si un nombre de usuario o un email están disponibles para un nuevo de usuario, es decir, todavía no se han registrado en el sistema.
- **UserProfile.** Se utiliza para devolver la información necesaria para mostrar la pantalla del perfil de un usuario. Contiene su identificador, su nombre de usuario, su nombre completo, foto de perfil y en qué fecha se ha registrado.
- **UserDetail.** Devuelve la misma información el *UserProfile* pero incluyendo los roles que tiene asignados el usuario.
- **ActivityDto.** Se utiliza en varios puntos. Este objeto contiene la representación de una actividad de entre los tres tipos existentes ya mencionados anteriormente. Contiene el usuario que hay hecho la acción y el que la recibe en caso de que aplique. Contiene la película a la que va dirigida la acción la actividad, de nuevo en caso de que aplique. Almacena la puntuación dada por un usuario a una película en caso que sea este tipo de actividad y la fecha en la que se ha realizado.
- **MoviePosterDto.** Se utiliza para devolver la imagen del poster de una película para poder mostrarla en pantalla en distintos puntos de la aplicación.

En el anexo C se explican los mapeos de las peticiones HTTP a los métodos Java de los controladores. Se explican solamente los más representativos para el funcionamiento general de la aplicación para que no se haga demasiado extenso.

7.2.3 Vista

Esta es la última capa de la arquitectura del sistema y se encarga de presentarle al usuario toda la información de la aplicación actualizada.

Al tratarse de una aplicación web, esta parte de la arquitectura se va a ejecutar dentro de un navegador web, por lo que en general se tratará de páginas *HTML* que son modificadas por código *JavaScript*. Pero por eficiencia y eficacia, como ya se ha adelantado, se va a utilizar un *framework* de alto nivel denominado *React.js*.

Al utilizar este *framework* se ha empledo el paradigma denominado programación orientada a componentes. Cada componente se encarga de representar una parte de la interfaz de usuario, por ejemplo, uno puede representar la barra superior y otro dentro del anterior un menú desplegable hacia abajo. Estos componentes se representan utilzando clases que heredan de *React.Component*. Todos los componentes deben implementar la función *render* que devuelva el contenido *HTML* que representa al componente. Esta función es llamada cada vez que se crea un componente o cuando su estado ha sido actualizado.

A continuación se expondrá brevemente la estructura general de un proyecto *react*. Dentro de la carpeta raiz de un proyecto existen tres directorios principales:

- **public/** este directorio es el directorio raiz del servidor. En el se encuentra el fichero *index.html*, que es el único fichero *HTML* necesario para el funcionamiento de la aplicación.
- **src/** es el directorio principal donde se defininarán todos los componentes y también otros ficheros necesarios, como llamadas a servicios.
- **node_modules/** es el directorio en el que se almacenan todas las dependencias con otras librerías.

Dentro del directorio src/ se encuentra el fichero *index.js* el cual es el punto de entrada del código *JavaScript* donde se inicializa el componente principal, *App.js*, para que se incruste en el *index.html*. A partir de aquí se ha definido dentro de esta misma carpeta todos los componentes necesarios. Estos se crean o bien directamente desde el componente principal o dentro de algún otro componente.

7.3 Integración con OMDb API

En este proyecto se ha decidido implementar la integración con la API OMDb (*The Open Movie Database*), este API permite conocer información de multitud de películas mantenida por sus usuarios.

Entre todos los datos que se pueden consultar, se ha decidido recuperar la información de la clasificación de una película concreta que tiene en otras webs dedicadas a contenido audivisual. Así el usuario puede consultar directamente desde la aplicación estas puntuaciones para tener una visión más global de la aceptación de la película por parte del mayor número de personas posible.

La integración que se ha implementado es bastante simple, cuando un usuario administrador da de alta una nueva película, este debe introducir también el identificador que le corresponde a la película en el sistema OMDb (*IMDb ID*). Posteriormente cuando sea necesario recuperarla información de una película, como puede ser cuando un usuario entra en el detalle de una, se realiza una llamada REST a la API y se recuperan también las puntuaciones de sistemas de terceros para mostrarlas lo más actualizadas posibles en el detalle de la película.

Para poder realizar esta llamada se ha tenido que registrar la aplicación en la web de la API para obtener la *API Key* que va como parámetro junto al identificador de la película en la petición que se realiza. La llamada a la API se realiza desde la parte servidora, así el usuario final no tiene acceso a esta *API Key*.

Capítulo 8

Implementación y pruebas

En este capítulo de la memoria se van a exponer los detalles del proceso que ha llevado a cabo para la codificación de la aplicación. Se explicará también la forma de compilar y ejecutar el proyecto.

8.1 Software requerido

Para llevar a cabo la codificación del proyecto se han utilizado las siguientes herramientas:

- **Eclipse IDE:** Versión Neon Release (4.6.0)
- **JDK:** Versión 1.8.0_92
- **Apache Maven:** Versión 3.3.9
- **MySQL:** Versión 14.14 Distrib. 5.7.13
- **Spring Boot:** Versión 2.0.3
- **Sublime Text:** Versión 3.1.1
- **Node.js:** Versión 8.11.3
- **NPM:** Versión 5.6.0
- **React:** Versión 16.4.1
- **Antd:** Versión 3.9.1
- **Bootstrap:** Versión 4.1.1
- **SQuirreL SQL Client:** Versión 3.8.1
- **Postman:** Versión 6.7.1

8.2 Estructura del proyecto

En lo relativo a la estructura del proyecto se debe dividir en dos partes bien diferenciadas. Por una parte está el código *Java* que se encarga de la lógica de negocio, mientras que por otra parte tenemos el código *React* (*JavaScript*) que se encarga de la parte de la interfaz de usuario.

8.2.1 Estructura proyecto Java

El proyecto se ha estructurado siguiendo el modelo estándar que se utiliza en los proyectos *Maven*. *Maven* se encarga de manejar las dependencias con otros *software* junto con sus versiones. Para configurar *Maven* se realiza en un fichero ”pom.xml” que se encuentra en el directorio raiz del proyecto. En este ficheros se declaran las dependencias además de definir la forma en la que se debe compilar y desplegar.

En la figura 8.1 se puede ver la jerarquía de carpetas y paquetes existentes en el desarollo del proyecto *Java*.

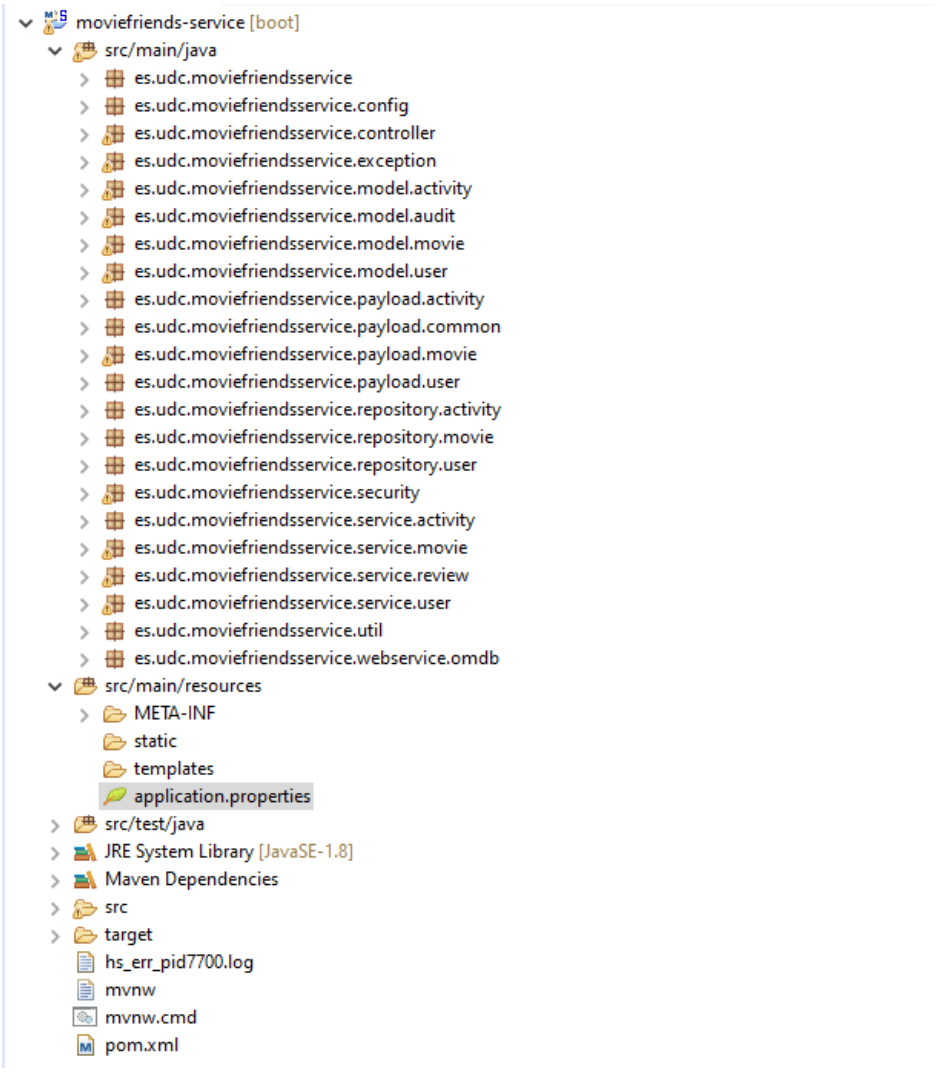


Figura 8.1: Estructura proyecto Java

En el directorio **src/main/java** se encuentra todo el código fuente de la parte controladora y modelo de la aplicación. Dentro, las clases *Java* están orginazadas en paquetes dependiendo de la función que cumplen, a continuación se explicarán brevemente los paquetes más importantes:

- **es.udc.moviefriendsservice**: Dentro de este paquete solamente se encuentra la clase *MoviefriendsServiceApplication.java* que es la que permite lanzar la aplicación *Spring Boot* de forma sencilla.
- **es.udc.moviefriendsservice.config**: En este paquete se encuentran las clases que permite configurar el contexto de *Spring*.

- **es.udc.moviefriendsservice.controller**: Paquete en el que están todas las clases que actúan como controladores dentro de la aplicación.
- **es.udc.moviefriendsservice.exception**: En este paquet se encuentran las excepcione utilizadas.
- **es.udc.moviefriendsservice.model.***: Paquetes en los que se guardan las clases que representan las entidades de base de datos.
- **es.udc.moviefriendsservice.payload.***: Paquetes en los que se almacenan las clases que se utilizan como objetos DTOs para transmitir información entre la interfaz de usuario y la lógica de negocio.
- **es.udc.moviefriendsservice.respository.***: Paquetes en los se guardan todos los respositories de la aplicación.
- **es.udc.moviefriendsservice.security**: Dentro de este paquete se guardan las clases que se utilizan en la funcionalidad de la autenticación de usuarios.
- **es.udc.moviefriendsservice.service.***: Paquetes en los que guardan las clases que actúan como servicios.
- **es.udc.moviefriendsservice.util**: Paquete en el que se almacenan clases de utilidad que implementan funciones de uso recuerrente en todo el código de la aplicación.
- **es.udc.moviefriendsservice.webservice.omdb**: Paquete que contiene las clases necesarias para la integración con OMDb API.

En el directorio **src/main/resources** solamente se guarda *application.properties* que define propiedades para configurar el entorno *Spring*.

8.2.2 Estructura proyecto React

El proyecto React se ha estructurado siguiendo la lógica de todos los proyectos React. Como ya se ha adelantado en la sección 7.2.3, un proyecto React se divide en tres directorios principales. El directorio public/ y node_modules/ no se han modificado, todo el código generado se ha guardado en **src/**. En la figura 8.2 se pueden ver los directorios del proyecto.

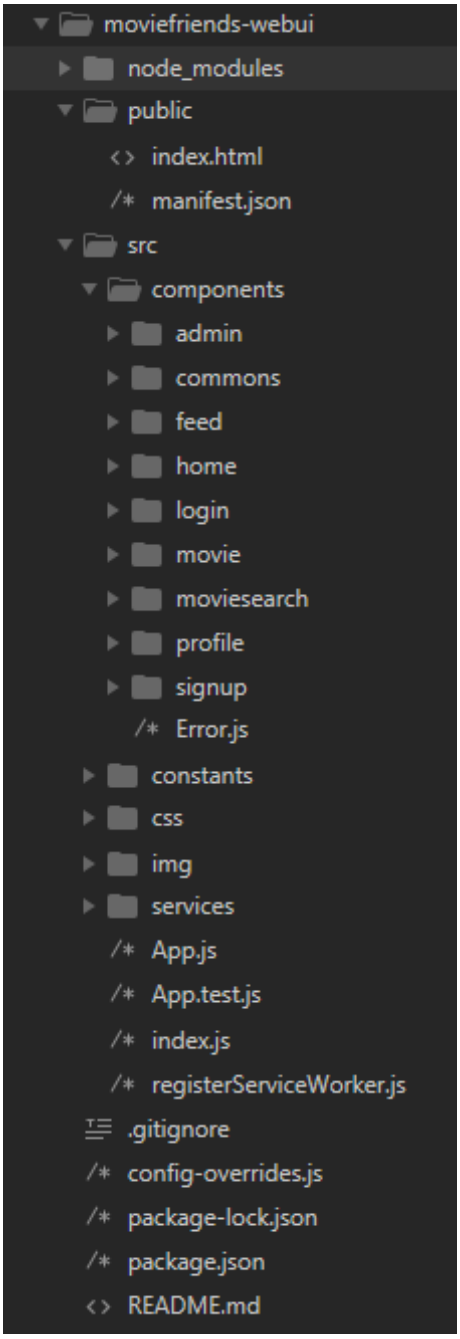


Figura 8.2: Estructura proyecto React

A continuación se explicará brevemente el contenido de cada directorio.

- **src/**: En este directorio se encuentra el fichero que define el componente principal *App.js* y el punto de entrada de *JavaScript* (*index.js*).

- **src/components**: Directorio en el que se almacenan todos los componentes separados en sus respectivos subdirectorios.
- **src/constants**: Directorio en el que se almacenan los ficheros con constantes definidas para poder usar en toda la aplicación.
- **src/css**: Directorio en el que se almacena el fichero CSS principal. Aparte, dentro de cada componente cada uno tiene su propio fichero CSS.
- **src/img**: Directorio en el que se almacenan las imágenes estáticas que se muestran en la aplicación.
- **src/services**: Directorio en el que se almacenan ficheros que implementan las funciones que permite realizar llamadas HTTP a los controladores de la capa modelo.

8.3 Instrucciones de compilación y ejecución

En esta sección se va a proceder a explicar paso por paso como compilar y ejecutar la aplicación. Antes que nada, es necesario avisar de que para que funcione correctamente es necesario que exista un servicio MySQL corriendo en la máquina local, con una base de datos creada con nombre ”moviefriends_app”. Además para autenticarse contra esta base de datos se utiliza el usuario ”root” con contraseña ”root”.

Primero se va a lanzar el proyecto Java que contiene el modelo de la aplicación.

1. Se abre una consola en el directorio raiz del proyecto (**moviefriends-service/**).
2. Se ejecuta el comando **mvn spring-boot:run**
3. Ya tenemos el corriendo la parte modelo de la aplicación en **localhost:5000**

Una vez ejecutada la parte modelo se va a lanzar la parte de la interfaz de usuario.

1. Se abre una consola en el directorio raiz del proyecto (**moviefriends-webui/**)
2. Se ejecuta el comando **npm start**
3. Abrir un navegador web en la dirección **http://localhost:3000**

8.4 Pruebas

En esta sección se van a explicar los aspectos más importantes del proceso de pruebas llevado a cabo durante el desarrollo de la aplicación. La fase de pruebas es muy importante dentro de un ciclo de desarrollo ya que permite asegurar que la implementación está libre de fallos y que además cumple con los requisitos acordados.

8.4.1 Pruebas unitarias

El objetivo de las pruebas unitarias es comprobar el correcto funcionamiento de elementos de código aislados, como puede ser una función Java. Idealmente estas pruebas deben de poder ejecutarse de forma aislada y deben poder repetirse las veces que se desee sin afectar a otras pruebas o al funcionamiento del sistema en general.

Estas pruebas han sido realizadas por el alumno de forma manual siguiendo una batería de pruebas para asegurar que el comportamiento de las funciones es correcto en todos los casos.

8.4.2 Pruebas de integración

Una vez realizadas las pruebas unitarias se deben llevar a cabo las pruebas de integración. Estas pruebas consisten en determinar si dos o más elementos de código aislado funcionan correctamente trabajando juntos. En general este tipo de pruebas se centran principalmente en la correcta comunicación entre componentes.

Estas pruebas han sido realizadas también por el alumno de forma manual como en las pruebas unitarias. Se ha seguido una batería con todos los casos.

8.4.3 Pruebas de sistema

El objetivo de las pruebas de sistema es comprobar el correcto funcionamiento de toda la aplicación en su conjunto, es decir, modelo, vista y controlador. Se han realizado utilizando un navegador web *Chrome*, navegando por todas las pantallas y validando que todas las funcionalidades se comportan de forma correcta.

8.4.4 Pruebas de aceptación

Este tipo de pruebas se realizan una vez finalizado todo el desarrollo y las pruebas anteriores para determinar si la aplicación resultante cumple o no los requisitos inicialmente

acordados. En una situación normal estas pruebas las realizaría el cliente, pero al tratarse de un trabajo de fin de grado las ha realizado el alumno junto con el director del proyecto.

Capítulo 9
Conclusiones

En este último capítulo se hara una reflexión sobre el resultado final del desarrollo realizado, además de posibles mejoras que se pueden realizar en un futuro.

9.1 Conclusiones

El sitema final cumple con todos los requisitos previamente acordados. Se ha desarrollado una aplicación web que permite puntuar, criticar y recomendar películas cuya interfaz gráfica se ha desarrollado utilizando la tecnología *React*. Esta interfaz es intuitiva, con una usabilidad alta y funciona en todos los navegadores modernos. Gracias a este desarrollo se han adquirido conocimientos y experiencia en una tecnología moderna de las más utilizadas en el mundo laboral.

También se ha diseñado e implementado una *API REST* siguiendo las guias de calidad propuestas por el estándar. Esta *API* es consumida por la parte vista para realizar operaciones y recuperar datos. Se ha realizado el desarollo utilizando *Java* y la tecnología *Spring Boot* que ha permitido minimizar mucho el tiempo de puesta en marcha del proyecto y en general ha facilitado el desarrollo. Para la parte de modelo de datos se ha diseñado e implementado una base de datos que encajara con toda la funcionalidad requerida para cumplir con todos los requisitos.

El objetivo general de este proyecto ha sido el desarrollo de una aplicación web lo más cercana al mundo real posible y además aprender del proceso llevado a cabo, y bajo mi punto vista este objetivo se ha cumplido.

9.2 Futuras líneas de trabajo

A continuación se exponen las posibles líneas de trabajo que se pueden seguir si se desea implementar nuevas funcionalidades o mejoras para aumentar el valor del prodcuto:

- Edición de atributos del perfil de usuario.
- Integración con diferentes APIs de redes sociales como *Facebook* o *Twitter*.
- Implementar un sistema de recomendación de películas basado en las críticas y puntuaciones realizadas por el usuario.
- Internacionalización a otros idiomas.
- Permitir a los usuarios administrador que puedan añadir actores, directores, guionistas y géneros de película desde la aplicación.

Apéndices

Apéndice A

Glosario de acrónimos

MVC *Model-View-Controller.*

JavaEE *Java Enterprise Edition*

API *Application Programming Interface*

IMDb *Internet Movie Database*

HTML *HyperText Markup Language*

ECMA *European Computer Manufacturers Association*

DOM *Document Object Model*

W3C *World WideWeb Consortium*

CSS *Cascading Style Sheets*

XHTML *eXtensible HyperText Markup Language*

IoC *Inversion of Control*

RMI *Remote Method Invocation*

AOP *Aspect Oriented Programming*

XML *eXtensible Markup Language*

CLI *Command-Line Interface*

SPA *Single-Page Application*

IDE *Integrated Development Environment*

IBM *International Business Machines Corporation*

POM *Project Object Module*

SQL *Structured Query Language*

JDBC *Java Database Connectivity*

GNU *General Public License*

OMDb *Open Movie Database*

JSON *JavaScript Object Notation*

REST *Representational State Transfer*

JPA *Java Persistence API*

HTTP *Hypertext Transfer Protocol*

RUP *Rational Unified Process*

BD *Base de Datos*

DTO *Data Transfer Object*

ID *Identifier*

CRUD *Create, Read, Update and Delete*

DSL *Domain Specific Language*

JPQL *Java Persistence Query Language*

JDK *Java Development Kit*

NPM *Node Package Manager*

Apéndice B

Glosario de términos

Streaming Es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se descarga

Feed Es un flujo de contenido por el que se puede desplazar. El contenido aparece en bloques parecidos que se repiten uno después del otro. Por ejemplo, un feed puede ser editorial (como una lista de artículos o noticias) o puede ser una ficha (una lista de productos, servicios, etc.).

Apéndice C

API Rest

AuthController

/api/auth/signin/	
Descripción	Permite la autenticación de un usuario en la aplicación.
Método	POST
Parametros	-
Payload	LoginRequest
Código respuesta	200 OK, 401 Unauthorized

Cuadro C.1: Autenticación de usuarios

/api/auth/signup/	
Descripción	Permite a un usuario registrarse en la aplicación.
Método	POST
Parametros	-
Payload	SignUpRequest
Código respuesta	201 Created, 400 Bad Request

Cuadro C.2: Registro de usuarios

AdminController

/api/admin/movies	
Descripción	Permite a un usuario administrador recuperar todas las películas registradas en el sistema de forma paginada.
Método	GET
Parametros	page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	201 Created, 403 Forbidden

Cuadro C.3: Lectura de todas las películas

/api/admin/movies	
Descripción	Permite a un usuario administrador registrar una película en la aplicación.
Método	POST
Parametros	-
Payload	RegisterMovieRequest
Código respuesta	201 Created, 400 Bad Request, 403 Forbidden

Cuadro C.4: Registro de películas

/api/admin/movies/{id}	
Descripción	Permite a un usuario administrador modificar una película registrada en el sistema.
Método	PUT
Parametros	id: identificador de la película.
Payload	ModifyMovieRequest
Código respuesta	201 Created, 400 Bad Request, 404 Not Found, 403 Forbidden

Cuadro C.5: Modificación de películas

/api/admin/movies/{id}	
Descripción	Permite a un usuario administrador eliminar una película registrada en el sistema.
Método	DELETE
Parametros	id: identificador de la película.
Payload	-
Código respuesta	204 No Content, 404 Not Found, 403 Forbidden

Cuadro C.6: Eliminación de películas

/api/admin/users	
Descripción	Permite a un usuario administrador recuperar todos los usuarios registrados en el sistema de forma paginada.
Método	GET
Parametros	page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	200 OK, 403 Forbidden

Cuadro C.7: Lectura de todos los usuarios

/api/admin/users	
Descripción	Permite a un usuario administrador registrar un usuario en el sistema.
Método	POST
Parametros	-
Payload	CreateUserByAdminRequest
Código respuesta	201 Created, 400 Bad Request, 403 Forbidden

Cuadro C.8: Registro de usuarios por un administrador

/api/admin/users/{id}	
Descripción	Permite a un usuario administrador modificar un usuario registrado en el sistema.
Método	PUT
Parametros	id: identificador del usuario
Payload	ModifyUserRequest
Código respuesta	201 Created, 400 Bad Request, 404 Not Found, 403 Forbidden

Cuadro C.9: Modificación de usuarios

/api/admin/users/{id}	
Descripción	Permite a un usuario administrador eliminar un usuario registrado en el sistema.
Método	DELETE
Parametros	id: identificador del usuario
Payload	-
Código respuesta	204 No Content, 404 Not Found, 403 Forbidden

Cuadro C.10: Eliminación de usuarios

/api/admin/actors	
Descripción	Permite a un usuario administrador recuperar todos los actores registrados en el sistema.
Método	GET
Parametros	-
Payload	-
Código respuesta	200 OK, 403 Forbidden

Cuadro C.11: Lectura de todos los actores

MovieController

/api/movies/search	
Descripción	Permite a un usuario realizar búsquedas con varios filtros.
Método	GET
Parametros	title: título de la película buscada. releaseYear: año de estreno de la película buscada. idsGenre: ids. de los géneros a los que pertenece la película buscada.
Payload	-
Código respuesta	200 OK, 400 Bad Request

Cuadro C.12: Búsqueda de películas con varios filtros.

/api/movies/{id}	
Descripción	Permite a un usuario recuperar toda la información detallada de una película registrada en el sistema.
Método	GET
Parametros	id: identificador de la película.
Payload	-
Código respuesta	200 OK, 404 Not Found

Cuadro C.13: Lectura del detalle de una película.

/api/movies/{id}/rate	
Descripción	Permite a un usuario puntuar una película.
Método	PUT
Parametros	id: identificador de la película.
Payload	RateMovieRequest
Código respuesta	200 OK, 404 Not Found

Cuadro C.14: Puntuar una película.

/api/movies/{id}/rate/exists	
Descripción	Permite a un usuario determinar si ya ha puntuado la película.
Método	GET
Parametros	id: identificador de la película.
Payload	-
Código respuesta	200 OK, 404 Not Found

Cuadro C.15: Determinar si un usuario ya ha puntuado una película.

/api/movies/mostviewed	
Descripción	Permite a un usuario recuperar de forma paginada una lista de películas ordenada por el número de votos que tiene.
Método	GET
Parametros	page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	200 OK

Cuadro C.16: Recuperar las películas más vistas.

/api/movies/topRated	
Descripción	Permite a un usuario recuperar de forma paginada una lista de películas ordenada por la puntuación media de votos que tiene.
Método	GET
Parametros	page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	200 OK

Cuadro C.17: Recuperar las películas más valoradas.

/api/movies/{id}/recommend	
Descripción	Permite a un usuario recomendar una película a otros usuarios.
Método	POST
Parametros	id: identificador de la película
Payload	RecommendMovieRequest
Código respuesta	200 OK, 404 Not Found

Cuadro C.18: Recomendar una película

/api/movies/{id}/reviews	
Descripción	Permite a un usuario recuperar todas las críticas que tiene una película de forma paginada.
Método	GET
Parametros	id: identificador de la película page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	200 OK, 404 Not Found, 403 Forbidden

Cuadro C.19: Recuperar las críticas de una película

/api/movies/{id}/reviews/exists	
Descripción	Permite a un usuario determinar si él mismo ya ha creado una crítica de una película.
Método	GET
Parametros	id: identificador de la película
Payload	-
Código respuesta	200 OK, 404 Not Found, 403 Forbidden

Cuadro C.20: Determinar si un usuario ya ha creado una crítica de una película

/api/movies/{id}/reviews	
Descripción	Permite a un usuario crear una crítica de una película.
Método	POST
Parametros	id: identificador de la película
Payload	RegisterReviewRequest
Código respuesta	201 Created, 404 Not Found, 403 Forbidden

Cuadro C.21: Crear una crítica de una película.

UserController

/api/users/checkUsernameAvailability	
Descripción	Permite a un usuario determinar si un nombre de usuario está disponible.
Método	GET
Parametros	username: nombre de usuario para comprobar
Payload	-
Código respuesta	200 OK

Cuadro C.22: Determinar si un nombre de usuario está disponible.

/api/users/checkEmailAvailability	
Descripción	Permite a un usuario determinar si un email de usuario está disponible.
Método	GET
Parametros	email: nombre de usuario para comprobar
Payload	-
Código respuesta	200 OK

Cuadro C.23: Determinar si un email de usuario está disponible.

/api/users/{username}	
Descripción	Permite a un usuario recuperar la información del perfil de un usuario.
Método	GET
Parametros	username: nombre de usuario
Payload	-
Código respuesta	200 OK, 404 Not Found

Cuadro C.24: Recuperar información del perfil de un usuario.

/api/users/{username}/follow	
Descripción	Permite a un usuario seguir a otro.
Método	POST
Parametros	username: nombre de usuario al que se va a empezar a seguir.
Payload	-
Código respuesta	200 OK, 404 Not Found, 409 Conflict

Cuadro C.25: Empezar a seguir a un usuario.

/api/users/{username}/unfollow	
Descripción	Permite a un usuario dejar de seguir a otro.
Método	POST
Parametros	username: nombre de usuario al que se va a dejar de seguir.
Payload	-
Código respuesta	200 OK, 404 Not Found, 409 Conflict

Cuadro C.26: Dejar de seguir a un usuario.

/api/users/{username}/checkUserFollowing	
Descripción	Permite a un usuario determinar si ya está siguiendo a otro usuario.
Método	GET
Parametros	username: nombre de usuario.
Payload	-
Código respuesta	200 OK, 404 Not Found

Cuadro C.27: Determinar si ya se está siguiendo a un usuario.

/api/users/me/feed	
Descripción	Permite a un usuario recuperar las actividades recientes de los usuarios a los que sigue. Es posible especificar el tipo de actividad que se quiere recuperar.
Método	GET
Parametros	typeOfActivity: identificador del tipo de actividad que se quiere recuperar. Es posible especificar que se quiere recuperar todos los tipos disponibles. page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	200 OK, 400 Bad Request

Cuadro C.28: Recuperar las actividades recientes de los usuarios seguidos.

/api/users/{username}/feed	
Descripción	Permite a un usuario recuperar las actividades recientes de un usuario concreto.
Método	GET
Parametros	typeOfActivity: identificador del tipo de actividad que se quiere recuperar. Es posible especificar que se quiere recuperar todos los tipos disponibles. username: nombre del usuario del que se quiere recuperar las actividades recientes. page: número de página. size: número de elementos de una página.
Payload	-
Código respuesta	200 OK, 400 Bad Request

Cuadro C.29: Recuperar las actividades recientes de un usuario concreto.

Bibliografía

[1] James Gosling, Bill Joy, Guy Steele, Gilad Bracha, and Alex Buckley. (2015) The java language specification java se 8 edition. [Online]. Available: https://docs.oracle.com/javase/specs/jls/se8/jls8.pdf

[2] E. Jendrock, R. Cervera-Navarro, I. Evans, K. Haase, and W. Markito, *The java ee 7 tutorial.* Addison-Wesley Professional, 2014, vol. 1.

[3] D. Flanagan, *JavaScript: the definitive guide.* " O'Reilly Media, Inc.", 2006.

[4] Javier Eguiluz. Introducción a css. [Online]. Available: https://uniwebsidad.com/libros/css

[5] Intorduction to latex. [Online]. Available: https://www.latex-project.org/about/

[6] R. Johnson, J. Hoeller, K. Donald, C. Sampaleanu, R. Harrop, T. Risberg, A. Arendsen, D. Davison, D. Kopylenko, M. Pollack *et al.*, "The spring framework–reference documentation," *interface*, 2004.

[7] C. Walls, *Spring Boot in action.* Manning Publications, 2016.

[8] C. Gackenheimer, *Introduction to React.* Apress, 2015.

[9] M. A. Jadhav, B. R. Sawant, and A. Deshmukh, "Single page application using angularjs," *International Journal of Computer Science and Information Technologies*, 2015.

[10] (2003) Eclipse platform technical overview. [Online]. Available: https://web.archive.org/web/20040716180615/http://www.eclipse.org/whitepapers/eclipse-overview.pdf

[11] Eclipse (software) - wikipedia. [Online]. Available: https://es.wikipedia.org/wiki/Eclipse_(software)

[12] E. Haughee, *Instant Sublime Text Starter.* Packt Publishing Ltd, 2013.

[13] R. Bharathan, *Apache Maven Cookbook.* Packt Publishing Ltd, 2015.

[14] R. Somasundaram, *Git: Version control for everyone*. Packt Publishing Ltd, 2013.

[15] Squirrel sql client home page - overview. [Online]. Available: http://www.squirrelsql.org/#overview

[16] About us - overleaf, editor de latex online. [Online]. Available: https://es.overleaf.com/about

[17] Postman - complete api development enviroment. [Online]. Available: https://www.getpostman.com/postman

[18] Mysql 8.0 reference manual - chapter 1: General information. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/introduction.html

[19] Omdb api web page. [Online]. Available: http://www.omdbapi.com/

[20] T. Berners-Lee, R. Fielding, and H. Frystyk, "Hypertext transfer protocol–http/1.0," Tech. Rep., 1996.

[21] D. Crockford, "The application/json media type for javascript object notation (json)," Tech. Rep., 2006.

[22] M. Masse, *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces.* " O'Reilly Media, Inc.", 2011.

[23] A. Leff and J. T. Rayfield, "Web-application development using the model/view/controller design pattern," in *Proceedings fifth ieee international enterprise distributed object computing conference*. IEEE, 2001, pp. 118–127.

[24] P. Kroll and P. Kruchten, *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional, 2003.

[25] M. A. Chaves, "La ingeniería de requerimientos y su importancia en el desarrollo de proyectos de software," *InterSedes*, vol. 6, no. 10, 2005.

[26] R. F. Grove and E. Ozkan, "The mvc-web design pattern." in *WEBIST*, 2011, pp. 127–130.

[27] The facade design pattern - problem, solution, and applicability - w3sdesign.com. [Online]. Available: http://w3sdesign.com/?gr=s05&ugr=proble

[28] The repository pattern - docs.microsoft.com. [Online]. Available: https://docs.microsoft.com/es-es/previous-versions/msp-n-p/ff649690(v=pandp.10)

[29] Data transfer object - docs.microsoft.com. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649585(v=pandp.10)

[30] A. Dearle, "Software deployment, past, present and future," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 269–284.

[31] Design patterns for optimizing the performance of j2ee applications - oracle.com. [Online]. Available: https://www.oracle.com/technetwork/articles/javaee/j2eepatterns-136748.html