ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Ingeniería del Software

Servicio multiplataforma para la consulta y gestión de contenido multimedia

Multiplatform service for consultation and manage multimedia content

Realizado por
José Antonio Palacios Ramírez
Tutorizado por
Eduardo Guzmán De los Riscos
Departamento
Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Diciembre 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: Hoy en día hay una gran cantidad de servicios que permiten la visualización de contenidos a través de internet, además de información referente a los mismos contenidos. Según un estudio, consumimos más de 8 horas semanales de contenido multimedia, ya sean series, películas, etc. Por ese motivo, tanto la información que consultamos como nuestra propia información generada en los propios servicios queda fragmentada. Este proyecto trata de solucionar este problema mediante el desarrollo de una aplicación multiplataforma para web y sistemas móviles Android e iOS. Su objetivo es centralizar toda la información de contenidos en esta aplicación mediante una experiencia de usuario sencilla y rápida de clasificación de películas en diferentes listas, que el usuario podrá gestionar desde su propio perfil. El sistema permitirá buscar películas por su título, seguir a otros usuarios y consultar la información relevante de la película, agregando además un amplio conjunto de puntuaciones de otras páginas importantes.
Palabras claves: Web, aplicación, Django, Django Rest Framework, HTML, CSS, Películas, Clasificación, Cine, bootstrap, PostgreSQL, API REST, IMDb, araña, Python.

Abstract: Nowadays, there are a a lot of services that allow the visualization of contents through the internet, even information about this content. According to a study, we consume more than 8 hours a week of multimedia content, like series, movies, etc. For this reason the information that we consult, and our own generated information on these services is fragmented. This project solves this problem through the development of a multiplatform application for the web and the mobile systems Android and iOS. It´s objective is to centralize all of this information in the application through a simple and fast experience for classifying movies in different lists, the user will be able to manage this information from his profile. The system will allow you to search for movies by title, you can follow other users and you can see the relevant information of the movie, including also a large set of ratings from other important pages.
Keywords: Web, application, Django, Django Rest Framework, HTML, CSS, movies, classification, cinema, bootstrap, PostgreSQL, API REST, IMDb, Scrapper, Python

Índice general

Capítulo 1. Introducción

1.1 Motivación

Internet ha propiciado que en estos días mucho de los contenidos y la información que visualizamos se consuma a través de aplicaciones web o móviles. En concreto, el consumo de contenidos digitales a la carta cada vez más se produce mediante plataformas de “Streaming” como Netflix, Wuaki … Antes del uso de dichas plataformas el usuario tenía que esperar a una hora en concreto para ver una serie o una película; con estos nuevos servicios se ha cambiado totalmente la forma y la demanda del contenido multimedia y la forma de consumo.

En 2011 los contenidos bajo demanda solo consumían tres horas de la semana del usuario, estos números aumentaron un 125% situándose en seis horas semanales. Los datos concretos como se indica en la figura 1 que nos muestra la evolución en estos años.

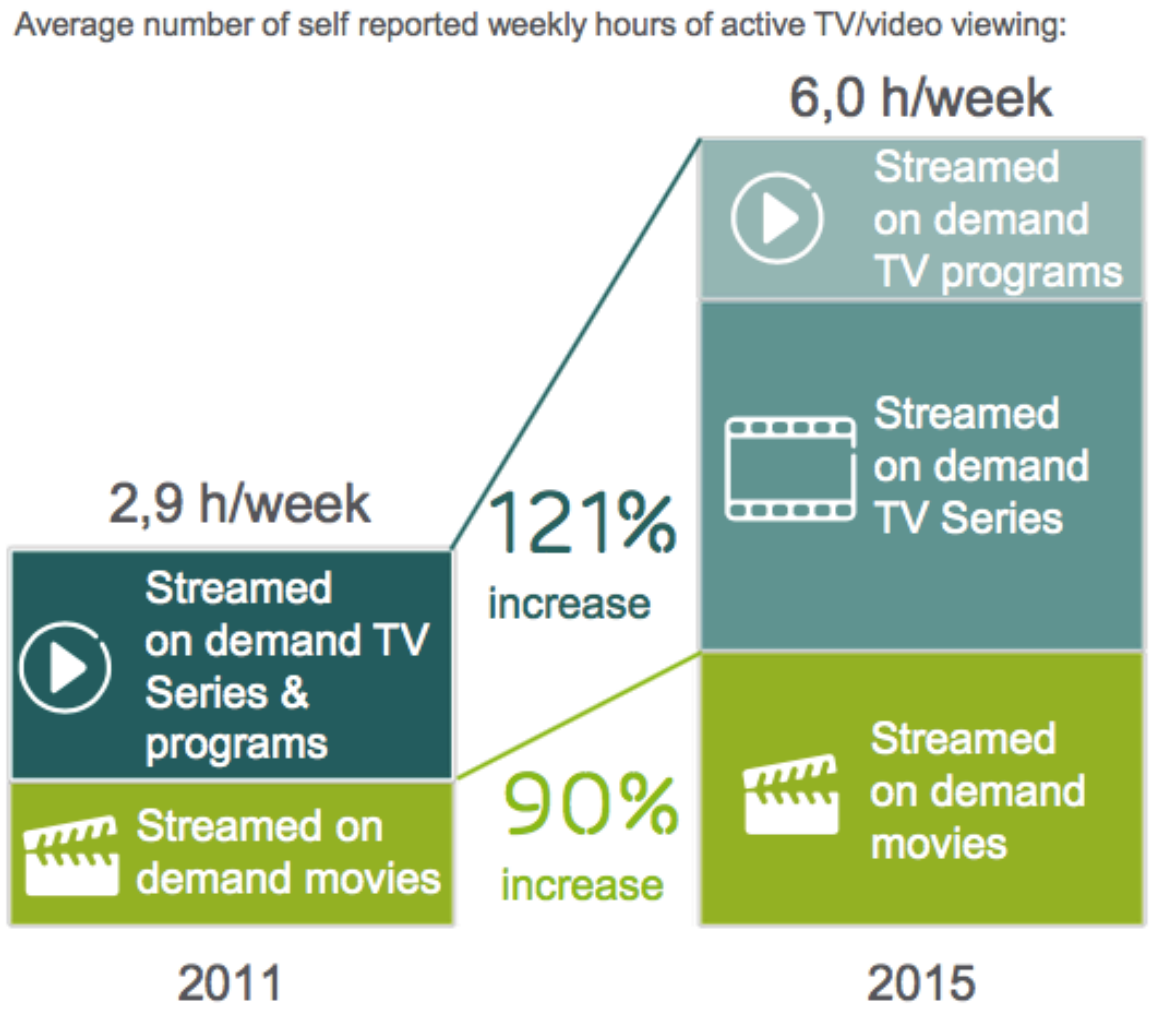

Figura 1.1. Datos de visionado.

Por otra parte, tenemos la multitud de páginas con información relevante de películas, sobre todo en el caso peculiar de las puntuaciones. Los usuarios toman como referencias estos valores a la hora de elegir su próxima película, como podemos ver en la siguiente figura.

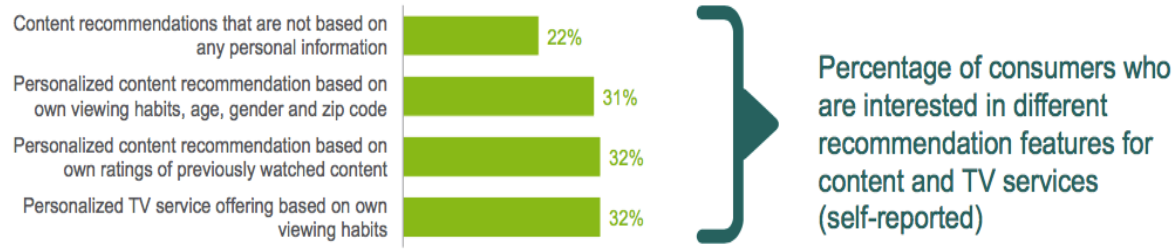

Figura 1.2. Datos relevantes para el visionado según los usuarios.

El 32% de las personas eligen el contenido en base a las puntuaciones, pero el principal problema es que estas suelen proporcionarlas diversos sitios web. Por esta razón, el usuario tiene que ir accediendo a cada uno de estos sitios para ver la puntuación asignada. Este proceso suele ser tedioso y lento, y requiere además, por parte del usuario, la comparación de las puntuaciones a la hora de determinar qué película elegir.

Ante esta situación, surge la necesidad de mantener el contenido centralizado en un mismo sitio. Además, haciendo esta gestión de tu contenido más simple y fácil de actualizar.

1.2 Objetivos

A raíz de estos problemas surge la necesidad de una aplicación multiplataforma con el objetivo de aportar a los usuarios una centralización de sus datos y una manera de clasificar su contenido de forma rápida y sencilla, accediendo también a la información relevante de la película como su duración, género, sinopsis o los actores que aparecen en ella. A parte de estas funcionalidades disponer de la app multiplataforma, para así acoger a un mayor público.

Por estas razones se ha desarrollado y documentado para este Trabajo de Fin de Grado una aplicación llamada Mooviest, pensada para que los usuarios puedan clasificar, buscar y consultar toda la información de películas, así como sus listas o de sus amigos. Para dotar al proyecto de más funcionalidad también realizamos unos scripts de automatización de tareas que se lanzarán temporalmente para mantener la aplicación siempre actualizada con las últimas películas y sus puntuaciones.

Para cumplir con los objetivos y funcionalidades se ha escogido el lenguaje de Django y un módulo del mismo que se llama Django Rest Framework, siendo ambos muy populares en creación de aplicaciones con desarrollo ágil y escalable.

Para la automatización y el almacenamiento se han usado algunos módulos de Python y la base de datos PostgreSQL, adaptándose muy bien a la estructura de datos relacional y a Django y su ORM.

1.3 Materiales y tecnología usada

- Hardware
    - Portatil Macbook Air 2013 con sistema operativo macOS Sierra
- Software
    - Editor de texto Atom
    - Navegador web Google Chrome y Safari
    - Administrador de paquetes PIP
    - Microsoft Word 2016
    - Magic Draw

1.4 Contenido de la memoria

Este trabajo de fin de grado se ha divido en varios capítulos abordando los temas más importantes de cada fase de desarrollo. Más concretamente, se ha seguido la siguiente estructura:

- Capítulo 1. Introducción: En este capítulo se introduce el proyecto a realizar indicando los motivos por los que surge la necesidad de la aplicación, cuáles son los objetivos del mismo, materiales utilizados para conseguirlo y este breve resumen del contenido de esta memoria.
- Capítulo 2. Conocimientos previos: En este capítulo introducimos al lector en las diferentes tecnologías utilizadas para llevar a cabo todo el desarrollo software con definiciones de cada una y nociones básicas.
- Capítulo 3. Especificación de requisitos: Tratamos los detalles sobre las funciones que debe cumplir la aplicación con un análisis de requisitos (funcionales y no funcionales) que marca el camino del desarrollo.
- Capítulo 4. Diseño: Entramos en la primera fase antes de comenzar el desarrollo, comentando los casos de uso que surgen tras el análisis de requisitos, además de una especificación de las diferentes arquitecturas y base de datos que se aplican.

- Capítulo 5. Implementación e instalación: Se trata de la sección con más extensión de este documento, ya que en ella se expondrá, de manera más descriptiva, el grueso del trabajo que, en este caso, es todo el desarrollo para construir la API y la aplicación Mooviest. Se comienza hablando de la estructura e implementación de la base de datos, creación y configuración del proyecto y, a partir de aquí, explicaremos en detalle cada sección de las aplicaciones, comentando primero los elementos visuales; tras esto explicaremos el funcionamiento de cada elemento.
- Capítulo 6. Conclusiones y trabajo futuro: Por último, resumiremos las conclusiones que se pueden extraer de toda la elaboración de un proyecto y se discutirán posibles trabajos futuros que se podrían realizar.
- Bibliografía: Material bibliográfico que se ha usado para el desarrollo de la aplicación y la realización de esta memoria.

1.5 División del trabajo

El proyecto se ha realizado en grupo con otros dos compañeros, consta de una parte común el servidor Django y una parte individual, en mi caso la aplicación Web. En la Tabla 1.1 podemos ver el porcentaje realizado por cada miembro.

			Jesús Garrido Moscoso	José Antonio Palacios Ramírez	Antonio Romero Gómez
Servidor Django	Creación e instalación		10%	80%	10%
	API	Movie	30%	40%	30%
		User	10%	80%	10%
	Scripts	Scrappers	45%	10%	45%
		Tviso	45%	10%	45%
		Trakt.tv	45%	10%	45%
		Script principal	40%	20%	40%
Aplicación Android			100%		
Aplicación IOS					100%
Aplicación Web				100%	

Tabla 1.1. División del Trabajo

Capítulo 2. Conocimientos previos

2.1 Python 3

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

Es un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Como comentábamos es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto compatible con la Licencia pública general de GNU. Fue creado a finales de los ochenta siendo Van Rossum su principal autor.

La versión 3 incluye una serie de cambios que hace que requiera reescribir el código de versiones anteriores.

2.2 Django

Django es un framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño de arquitecturas Modelo-Vista-Controlador. La meta de Django es la creación de sitios webs complejos, de forma ágil y fácil, poniendo énfasis en la reutilización, conectividad y extensibilidad de componentes, con el principal principio “no te repitas” (DRY, *Don´t Repeat Yourself*).

2.3 Django Rest Framework

Es una librería que nos permite construir un API REST sobre Django de forma sencilla. Ofrece una gran variedad de método y funciones para el manejo, definición y control de nuestros recursos.

Incluye varios aspectos importantes en el diseño y creación de APIs tales como la autenticación.

Los conceptos más usados durante la creación de nuestra API REST son los siguientes:

- Serializadores: Permite que los datos complejos, como consultas e instancias a tipos de datos nativos de Python puedan ser fácilmente transmitidos en otro tipo de datos como JSON o XML y viceversa, obtener un tipo de dato y convertirlo en datos complejos de Python.
- Viewsets: Permiten combinar la lógica de un conjunto de controladores relacionados en una sola clase. En el viewset se realiza la manipulación de las peticiones GET y POST y creación de los diferentes puntos de entradas de la API

2.4 PostgreSQL

Fundado en 1982 con el proyecto Ingres en la Universidad de Berkeley, es un sistema de gestión de bases de datos relacional orientado a objetos y libre. Está dirigido por una comunidad de desarrolladores bajo licencia PostgreSQL, similar a la BSD o a la MIT. Sus características:

- Alta concurrencia: Permite que mientras un proceso realiza una escritura en una tabla, otros accedan a la misma sin necesidad de bloqueos.
- Amplia variedad de tipos nativos.
- Funciones: Bloques de código que se ejecutan en el servidor. Pueden ser escritos en varios lenguajes, con la potencia de cada uno de ellos da, tales como operaciones básicas, bifurcaciones o bucles.

Otra característica menos conocida son las notificaciones en tiempo real con las funciones LISTEN, UNLISTEN y NOTIFY.

2.5 API REST

Es un tipo de arquitectura de desarrollo web que se apoya en el estándar HTTP. Nos permite crear servicios y aplicaciones que pueden ser usadas en cualquier dispositivo o cliente que entienda HTTP.

Como hemos comentado, trabaja con HTTP y sus métodos que son:

- GET: Para consultar y leer recursos.
- POST: Para crear recursos.
- PUT: Para actualizar recursos.
- DELETE: Para eliminar recursos.
- PATCH: Para actualizar partes concretas de los recursos.

Además, parte importante de la creación de una API REST son los códigos de estado para saber en qué punto o por qué falla la petición aún recurso. Algunos de ellos son:

- Error 500: Fallo del servidor.
- Error 404: Recurso no encontrado.
- Error 200: Petición realizada correctamente.

2.6 HTML

HTML (Hyper Text Mark-up Language) es un lenguaje de marcado básicamente usado para la creación de páginas web. Se usa para escribir y traducir la estructura e información en forma de texto, así como ilustrar el texto con elementos como imágenes, vídeos, etc.

El código se define mediante etiquetas (<inicio></fin>) que permiten añadir cualquier elemento a una página web. También permiten incluir scripts de otros lenguajes de programación como JavaScript.

2.6 CSS

CSS (Cascade Style Sheets, Hojas de estilo en cascada) es un lenguaje formal utilizado para definir la presentación de un documento estructurado escrito en HTML.

La idea principal del uso de las hojas de CSS es separar la estructura de un documento de su presentación. Cuando se utiliza CSS, las etiquetas de los documentos HTML no facilitan la información del formato de la presentación, sino que marcan la estructura. CSS se encarga de concretar cómo se visualiza el contenido dentro de la etiqueta: color, fuente, posición …

2.6 Bootstrap

Es un Framework o conjunto de herramientas de código abierto para diseño de aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, etc. Basado en HTML y CSS, así como, extensiones de JavaScript opcionales.

2.7 pip (Python Package Index)

Es un sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python.

Capítulo 3. Especificación de requisitos

La especificación de requisitos de un sistema consiste en una descripción del comportamiento que ha de implementar este sistema y las restricciones asociadas a su funcionamiento. Esta descripción se clasifica habitualmente en dos grupos:

- Requisitos funcionales: Es el tipo más común. Identifican los servicios o comportamientos que el sistema debe ofrecer, normalmente en respuesta a una entrada externa.
- Requisitos no funcionales: Este tipo por otra parte nos especifica las restricciones en el diseño e implementación, es decir, restricciones que afectaran al correcto funcionamiento del sistema.

A continuación, vamos a comentar la recopilación de información que se ha llevado a cabo para, seguidamente, extraer los requisitos en base a ella.

3.1 Recopilación de datos

Al tratarse de un Trabajo de Fin de Grado en el que desarrollamos una idea propia, hicimos un trabajo de estudio inicial analizando qué características tenían otras aplicaciones similares y qué información mostraban en el ámbito de películas. Además estudiamos la documentación existente sobre APIs públicas para poder almacenar todos los datos en nuestra base de datos. A través de correo electrónico contactamos con los responsables de varias de estas aplicaciones para pedir permiso en cuanto al uso de algunos datos. Como resultado, desde muchas de estas aplicaciones se nos permitió el acceso. Además nos cedieron imágenes y nos proporcionaron algunas pautas de diseño y colores de sus logos.

Después de esta fase inicial, y tras conseguir todo lo necesario, realizamos una reunión de tormenta de ideas para definir de mejor manera qué funcionamiento tendría nuestra aplicación y qué elementos diferenciadores podríamos aportar respecto a las otras aplicaciones.

3.2 Aplicación web

3.2.1 Requisitos funcionales

- **RF01 – Iniciar sesión:** El usuario podrá iniciar sesión introduciendo su correo electrónico o nombre de usuario y su contraseña.
    - **RF01.1 – Introducir nombre de usuario o correo electrónico:** El usuario deberá introducir su nombre de usuario o correo electrónico.
    - **RF01.2 – Introducir contraseña:** El usuario deberá introducir su contraseña.
    - **RF01.3 – Error:** Si ha introducido alguno de los datos y es incorrecto se mostrará un texto con el error.
- **RF02 – Registrarse:** El usuario podrá registrarse en la aplicación indicando todos los datos necesarios para completar el registro.
    - **RF02.1 Introducir nombre de usuario:** El usuario deberá introducir el nombre de usuario que el elija.
    - **RF02.2 Introducir email:** El usuario deberá introducir el nombre de usuario que el elija.
    - **RF02.3 Introducir contraseña:** El usuario deberá introducir la contraseña que el elija.

    - **RF02.4 Error:** Si no ha introducido alguno de los datos o alguno no es correcto, entonces se mostrará *n* mensaje de error con lo que ha ocurrido.

- **RF03 – Mostrar menú superior:** La aplicación deberá mostrar siempre un menú en la parte superior de la página con todas las secciones de la aplicación para mejorar la navegabilidad.
- **RF04 – Mostrar Página inicial:** La aplicación deberá tener una página inicial donde mostrará las últimas películas clasificadas.
    - **RF04.1 Pendientes:** Se presentarán en una fila las últimas películas clasificadas como pendientes.
    - **RF04.2 Favoritas:** Se presentarán en una fila las últimas películas clasificadas como pendientes.
    - **RF04.2 Vistas:** Se presentarán en una fila las últimas películas clasificadas como vistas.
    - **RF04.2 Lista negra:** Se presentarán en una fila las últimas películas clasificadas como lista negra.
- **RF05 – Clasificar películas:** El usuario podrá clasificar la película pulsando diferentes botones en la página de detalle de la película.
    - **RF05.1 – Clasificar película como Favorita:** El usuario podrá pulsar un botón para indicar que esa película es su favorita.
    - **RF05.2 – Clasificar película como Pendiente:** El usuario podrá pulsar un botón para indicar que esa película la tiene pendiente para ver.
    - **RF05.2 – Clasificar película como Vista:** El usuario podrá pulsar un botón para indicar que esa película la tiene vista.
    - **RF05.2 – Clasificar película como no me interesa:** El usuario podrá pulsar un botón para indicar que esa película no le interesa.
- **RF06 – Consultar perfil:** El usuario podrá ver su perfil y las diferentes listas que tiene y personas relacionadas con él.
    - **RF06.1 Consultar Lista de Pendientes:** El usuario podrá ver su lista de películas clasificadas en pendientes.
    - **RF06.2 Consultar Lista de Vistas:** El usuario podrá ver su lista de películas clasificadas como vistas.
    - **RF06.3 Consultar Lista de Lista negra:** El usuario podrá ver su lista de películas clasificadas en su lista negra.

    - **RF06.4 Consultar Lista de Favoritos:** El usuario podrá ver su lista de películas clasificadas en favoritas.
    - **RF06.5 Consultar seguidores:** El usuario podrá ver a todas las personas que le siguen.
    - **RF06.6 Consultar seguidos:** El usuario podrá ver todas las personas que ha seguido.
- **RF07 – Consultar cuenta:** El usuario podrá visualizar sus datos como correo electrónico, nombre de usuario, género, ciudad, país, su foto, código postal, fecha de nacimiento, nombre y apellidos.
- **RF07 – Modificar cuenta:** El usuario podrá modificar todos los datos referentes a su cuenta como nombre de usuario, correo electrónico, nombre, apellidos, foto, fecha de nacimiento, género, país, ciudad, código postal o contraseña.
- **RF08 – Cerrar sesión:** El usuario podrá cerrar sesión.
- **RF09 – Consultar detalle de una película:** El usuario podrá ver toda la información detallada sobre esa película.
    - **RF09.1 Comparativa de puntuaciones:** El usuario podrá ver una comparativa de todas las puntuaciones de diferentes sitios web.
    - **RF09.2. Reparto:** El usuario podrá ver todas las personas involucradas con la película, tales como actores, escritores, directores …
    - **RF09.3 Consultar género:** El usuario podrá consultar los diferentes géneros en los que se puede clasificar la película.
    - **RF09.4 Duración:** El usuario podrá consultar la duración de la película.
    - **RF09.5 Error:** Si no encuentra una película, se lanzará una página con el mensaje no encontrado.
- **RF10 – Buscar películas:** El usuario podrá buscar películas tanto por su título original como por su traducción en el idioma de su perfil.
    - **RF10.1 Error**: Si no encuentra ninguna película entonces saldrá un mensaje con un mensaje.
- **RF11 – Buscar usuarios**: El usuario podrá buscar a otros usuarios por su nombre completo o por su nombre de usuario.
- **RF12 – Consultar otros perfiles:** El usuario podrá consultar el perfil de otros usuarios.
    - **RF012.1 Consultar Lista de Pendientes:** El usuario podrá ver su lista de películas clasificadas en pendientes por el otro usuario.

    - **RF12.2 Consultar Lista de Vistas:** El usuario podrá ver su lista de películas clasificadas como vistas por el otro usuario.
    - **RF12.3 Consultar Lista de Lista negra:** El usuario podrá ver su lista de películas clasificadas en su lista negra por el otro usuario.
    - **RF12.4 Consultar Lista de Favoritos:** El usuario podrá ver la lista de películas clasificadas en favoritas por el otro usuario.
    - **RF12.5 Consultar seguidores:** El usuario podrá ver a todas las personas que le siguen.
    - **RF12.6 Consultar seguidos:** El usuario podrá ver todas las personas que sigue.
- **RF13 – Seguir usuarios:** El usuario podrá seguir a otros usuarios.
- **RF14 – Dejar de seguir:** El usuario podrá dejar de seguir a otros usuarios.
- **RF15 – Soporte multilenguaje:** El usuario tendrá la aplicación en el idioma de su navegador.

3.2.2 Requisitos no funcionales

- **RFN01 – Conexión a internet:** La aplicación necesita conexión a internet para el acceso a la aplicación y la base de datos.
- **RFN02 – Cambio de contraseña:** Solo puede ser modificada por el propio usuario o el administrador.

Capítulo 4. Diseño

4.1 Casos de uso

4.1.1 Aplicación Mooviest (Web)

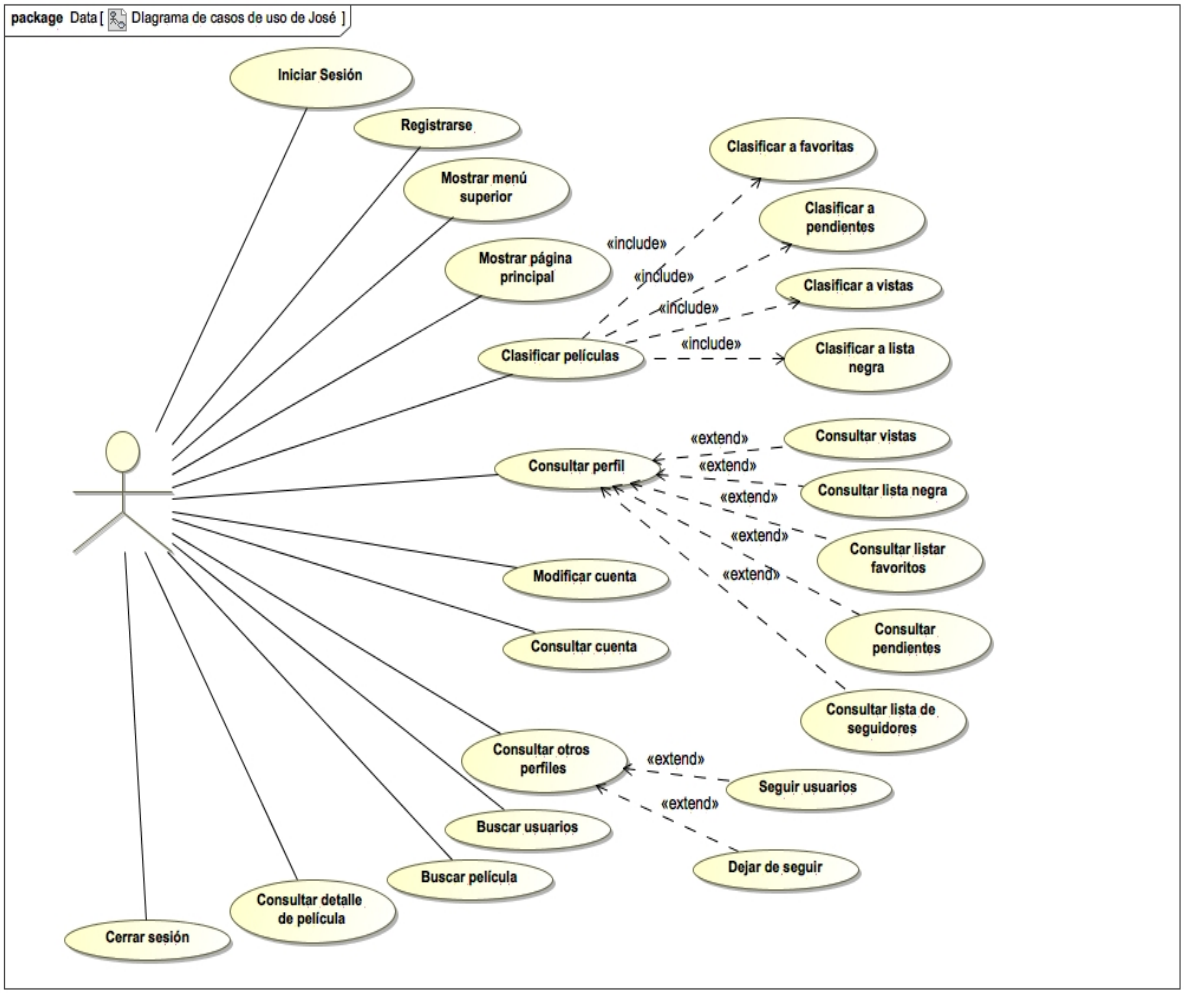

Figura 4.1. Casos de uso.

CU01	**Iniciar sesión**	
Descripción	El usuario inicia sesión en la aplicación	
Precondición	El usuario ha abierto el navegador y ha introducido en enlace de la aplicación.	
Escenario principal	**Paso**	**Acción**
	1	El usuario hace clic en “Iniciar sesión”
	2	El sistema carga la página de inicio de sesión

	3	El usuario introduce su email o nombre de usuario
	4	El usuario introduce su contraseña
	5	El usuario pulsa el botón “INICIAR SESIÓN”
	6	El sistema comprueba que es un usuario existente
	7	El sistema obtiene los datos del usuario
	8	El sistema carga la página principal de la aplicación (*Home*)
Postcondición	El usuario ha iniciado sesión en la aplicación	
Escenario alternativo	**Paso**	**Acción**
	6	Si el usuario no existe en el sistema
	E1	El sistema informa que el usuario no existe en la aplicación

CU02	**Registrarse**	
Descripción	El usuario se registra en la aplicación	
Precondición	El usuario ha abierto el navegador y ha introducido en enlace de la aplicación.	
Escenario principal	**Paso**	**Acción**
	1	El usuario ha hecho clic en “Regístrate”
	2	El sistema carga la página de registro del usuario
	3	El usuario introduce su nombre de usuario
	4	El usuario introduce su email
	5	El usuario introduce una contraseña
	6	El usuario confirma la contraseña
	7	El usuario pulsa el botón “Regístrate”
	8	El sistema comprueba que los datos son válidos
	9	El sistema registra al usuario en la aplicación
	10	El sistema carga la página principal de la aplicación
Postcondición	El usuario ha sido registrado en el sistema	
	Paso	**Acción**

Escenario alternativo	8	Si el usuario no existe en el sistema	
		E1	El sistema informa que el usuario no existe en la aplicación
		E2	El sistema informa que el usuario ya existe en la aplicación

CU03	**Mostrar menú superior**	
Descripción	El usuario podrá visualizar el menú superior de la aplicación	
Precondición	El usuario ha abierto el navegador y ha introducido en enlace de la aplicación.	
Escenario principal	**Paso**	**Acción**
	1	El sistema carga el menú superior y la página principal
Postcondición	El usuario ve el menú superior	
Escenario alternativo	**Paso**	**Acción**

CU04	**Mostrar Página principal**	
Descripción	El usuario podrá visualizar la página principal de la aplicación	
Precondición	El usuario ha abierto el navegador y ha introducido en enlace de la aplicación.	
Escenario principal	**Paso**	**Acción**
	1	El sistema carga el menú superior
	2	El sistema carga las últimas películas clasificadas como pendientes
	3	El sistema carga las últimas películas clasificadas como favoritas
	4	El sistema carga las últimas películas clasificadas como vistas
	5	El sistema carga las últimas películas clasificadas como lista negra
Postcondición	El usuario visualiza la página principal	
Escenario alternativo	**Paso**	**Acción**

CU05	Clasificar películas	
Descripción	El usuario podrá consultar el sistema de clasificar películas.	
Precondición	CU01 ó CU02	
Escenario principal	Paso	Acción
Postcondición	El usuario consulta el sistema de clasificación de películas	
Escenario alternativo	Paso	Acción

CU06	Clasificar una película como favoritas	
Descripción	El usuario podrá añadir una película a su lista de favoritos, pulsando el botón de favorita.	
Precondición	CU01 ó CU02	
Escenario principal	Paso	Acción
	1	El usuario pulsa el botón con el icono de la estrella.
	2	El sistema añade la película a su lista de favoritas
	3	El sistema actualiza el botón.
Postcondición	El usuario ha clasificado una película como favoritas	
Escenario alternativo	Paso	Acción
CU07	Clasificar una película como pendientes	
Descripción	El usuario podrá añadir una película a su lista de pendientes pulsando el botón correspondiente.	
Precondición	CU01 ó CU02	
Escenario principal	Paso	Acción
	1	El usuario pulsa el botón con el icono del marca páginas.
	2	El sistema añade la película a su lista de pendientes.
	3	El sistema actualiza el botón.
Postcondición	El usuario ha añadido una película a su lista de pendientes.	

Escenario alternativo	Paso	Acción

CU08	Clasificar una película como vista	
Descripción	El usuario podrá añadir una película a su lista de vistas, pulsando el botón de vista.	
Precondición	CU01 ó CU02	
Escenario principal	Paso	Acción
	1	El usuario pulsa el botón con el icono del ojo.
	2	El sistema añade la película a la lista de vistas.
	3	El sistema actualiza el botón.
Postcondición	El usuario ha añadido la película a su lista de vistas	
Escenario alternativo	Paso	Acción

CU09	Añadir una película a no me interesa	
Descripción	El usuario podrá añadir una película a su lista negra pulsando el botón correspondiente.	
Precondición	CU01 ó CU02	
Escenario principal	Paso	Acción
	1	El usuario pulsa el botón con el icono de la X.
	2	El sistema añade la película a no me interesa.
	3	El sistema actualiza el botón.
Postcondición	El usuario ha añadido una película a no me interesa.	
Escenario alternativo	Paso	Acción

CU10	Consultar perfil	
Descripción	El usuario podrá consultar su perfil	
Precondición	CU01 ó CU02, CU03	
	Paso	Acción

Escenario principal	1	El usuario pulsa sobre el menú desplegable.
	2	El usuario pincha en el enlace “Perfil”
	3	El sistema carga la página del perfil de usuario
Postcondición	El usuario consulta su perfil.	
Escenario alternativo	**Paso**	**Acción**

CU11	**Consultar lista de pendientes**	
Descripción	El usuario podrá consultar su lista de pendientes	
Precondición	CU01 ó CU02, CU03, CU10	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre el enlace “Listas”
	2	El sistema despliega un submenú con las listas
	3	El usuario pulsa sobre el enlace de “Pendientes”
	4	El sistema carga la lista de películas pendientes
Postcondición	El usuario consulta su lista de pendientes.	
Escenario alternativo	**Paso**	**Acción**

CU12	**Consultar lista de vistas**	
Descripción	El usuario podrá consultar su lista de películas vistas	
Precondición	CU01 ó CU02, CU03, CU10	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre el enlace “Listas”
	2	El sistema despliega un submenú con las listas
	3	El usuario pulsa sobre el enlace de “Vistas”
	4	El sistema carga la lista de películas vistas
Postcondición	El usuario consulta su lista de vistas.	
Escenario alternativo	**Paso**	**Acción**

CU13	Consultar lista no me interesa	
Descripción	El usuario podrá consultar su lista de películas que no le interesan.	
Precondición	CU01 ó CU02, CU03 ,CU10	
Escenario principal	Paso	Acción
	1	El usuario pulsa sobre el enlace “Listas”
	2	El sistema despliega un submenú con las listas
	3	El usuario pulsa sobre el enlace de “Lista negra”
	4	El sistema carga la lista de películas en su lista negra
Postcondición	El usuario consulta su lista de películas en lista negra.	
Escenario alternativo	Paso	Acción

CU14	Consultar lista de favoritos	
Descripción	El usuario podrá consultar su lista de favoritos	
Precondición	CU01 ó CU02, CU03, CU10	
Escenario principal	Paso	Acción
	1	El usuario pulsa sobre el enlace “Listas”
	2	El sistema despliega un submenú con las listas
	3	El usuario pulsa sobre el enlace de “Favoritas”
	4	El sistema carga la lista de películas favoritas
Postcondición	El usuario consulta su lista de favoritas.	
Escenario alternativo	Paso	Acción

CU15	Consultar lista de seguidores	
Descripción	El usuario podrá consultar su lista de seguidores	
Precondición	CU01 ó CU02, CU03, CU10	
Escenario principal	Paso	Acción
	1	El usuario pulsa sobre el enlace “Seguidores”

	2	El sistema carga la lista de usuarios seguidores.
Postcondición	El usuario consulta su lista de seguidores.	
Escenario alternativo	**Paso**	**Acción**

CU16	**Consultar lista de seguidos**	
Descripción	El usuario podrá consultar su lista de seguidos	
Precondición	CU01 ó CU02, CU03, CU10	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre el enlace “Seguidos”
	2	El sistema carga la lista de usuarios seguidos.
Postcondición	El usuario consulta su lista de usuarios seguidos.	
Escenario alternativo	**Paso**	**Acción**

CU17	**Consultar cuenta**	
Descripción	El usuario podrá consultar su cuenta	
Precondición	CU01 ó CU02, CU03	
Escenario principal	**Paso**	**Acción**
	1	El usuario navega a la página de ajustes.
	2	El sistema carga los datos del perfil de usuario
Postcondición	El usuario consulta su cuenta.	
Escenario alternativo	**Paso**	**Acción**

CU18	**Modificar Cuenta**	
Descripción	El usuario podrá consultar su perfil	
Precondición	CU01 ó CU02, CU03, CU17	
Escenario principal	**Paso**	**Acción**
	1	El usuario modifica los datos del perfil de usuario.

	2	El usuario pulsa el botón “Guardar”	
	3	El sistema comprueba que los datos son válidos.	
	4	El sistema guarda los datos del usuario.	
	5	El sistema carga la página principal.	
Postcondición	El usuario ha modificado su perfil.		
Escenario alternativo	**Paso**	**Acción**	
	3	Comprueba los datos introducidos	
		E1	El sistema comprueba datos requeridos
		E2	El sistema informa que no se puede repetir ese nombre de usuario

CU19	**Cerrar sesión**	
Descripción	El usuario podrá cerrar sesión.	
Precondición	CU01 ó CU02, CU03	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre el menú desplegable
	2	El usuario pulsa el enlace “Cerrar sesión”
	3	El sistema borra los datos de sesión.
	4	El sistema carga la página principal.
Postcondición	El usuario ha cerrado sesión.	
Escenario alternativo	**Paso**	**Acción**

CU20	**Consultar detalle de una película**	
Descripción	El usuario podrá consultar el detalle de una película	
Precondición	CU01 ó CU02	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre la caratula de alguna película
	2	El sistema carga toda la información relevante de películas.

	3	El sistema muestra toda la información de película como puntuaciones, reparto, género …	
Postcondición	El usuario ha visualizado el detalle de una película		
Escenario alternativo	**Paso**	**Acción**	
	3	La página de detalle no existe	
		E1	Mostrará una página con error “No encontrado”

CU21	**Buscar películas**		
Descripción	El usuario podrá buscar películas		
Precondición	Aplicación abierta		
Escenario principal	**Paso**	**Acción**	
	1	El usuario pulsa sobre la caja de texto del buscador.	
	2	El usuario introduce un título	
	3	El usuario pulsa sobre el botón con una lupa.	
	4	El sistema busca películas que cumplan las condiciones de búsqueda.	
	5	El sistema carga una página con las películas encontradas.	
Postcondición	El usuario visualiza las películas encontradas.		
Escenario alternativo	**Paso**	**Acción**	
	3	Si el sistema no encuentra resultados	
		E1	El sistema informa que no ha encontrado resultados.

CU22	**Buscar usuarios**	
Descripción	El usuario podrá buscar otros usuarios	
Precondición	CU01 O CU02	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre la caja de texto del buscador.
	2	El usuario introduce un nombre de usuario o nombre y apellidos.

	3	El usuario pulsa sobre el botón con una lupa.	
	4	El sistema busca usuarios que cumplan las condiciones de búsqueda.	
	5	El sistema carga una página con los usuarios encontrados.	
Postcondición	El usuario visualiza los usuarios encontrados.		
Escenario alternativo	**Paso**	**Acción**	
	3	Si el sistema no encuentra resultados	
		E1	El sistema informa que no ha encontrado resultados.

CU23	**Consultar otros perfiles**		
Descripción	El usuario podrá consultar el perfil de otros usuarios.		
Precondición	CU01 O CU02, CU22		
Escenario principal	**Paso**	**Acción**	
	1	El sistema carga la información del usuario.	
	2	El usuario puede navegar pulsando cualquier enlace como listas, seguidores o seguidos de ese usuario.	
	3	El sistema mostrará toda esa información en forma de lista.	
Postcondición	El usuario visualiza los usuarios encontrados.		
Escenario alternativo	**Paso**	**Acción**	
	3	Si el sistema no encuentra resultados	
		E1	El sistema informa que no ha encontrado resultados.

CU24	**Seguir usuarios**
Descripción	El usuario podrá seguir a otro usuario
Precondición	CU01 O CU02, CU23
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre el botón seguir
	2	El sistema actualiza el botón.

	3	El sistema guarda el usuario como seguido.
Postcondición	El usuario ha seguido a otro usuario	
Escenario alternativo	**Paso**	**Acción**

CU25	**Dejar de seguir**	
Descripción	El usuario podrá dejar de seguir a otro usuario	
Precondición	CU01 O CU02, CU23, CU24	
Escenario principal	**Paso**	**Acción**
	1	El usuario pulsa sobre el “Dejar de seguir”
	2	El sistema actualiza el botón.
	3	El sistema actualiza el contenido de la lista de seguidos.
Postcondición	El usuario ha dejado de seguir a otro usuario.	
Escenario alternativo	**Paso**	**Acción**

4.2 Arquitectura

La arquitectura software es el diseño a más alto nivel de la estructura de un sistema. Este tipo de arquitecturas ha ido surgiendo para simplificar problemas generales indicando estructura, funcionamiento e interacción entre las partes.

En base a nuestros requerimientos y restricciones podemos distinguir dos tipos de arquitecturas en este proyecto: por un lado la arquitectura a alto nivel,cliente/servidor en la que tendremos como clientes las aplicaciones móviles y web. El rol de servidor lo desempeña nuestra API con Django Rest Framework. Adicionalmente, usaremos el patrón de diseño de arquitecturas, Modelo-Vista-Controlador, para el desarrollo de la aplicación en Django.

4.2.1 Arquitectura cliente-servidor

Es un modelo de aplicación distribuida donde encontramos el cliente y servidor aislados uno de otro y conectados a través de internet. Esta conexión solo se realiza cuando el cliente comienza la comunicación mediante una llamada al servidor; el servidor responderá, pero nunca podrá iniciar esta comunicación.

Por lo tanto, nuestro servidor nos suministra un servicio: darnos la información de la base de datos o realizar las acciones de computación para cuando el cliente lo solicite.

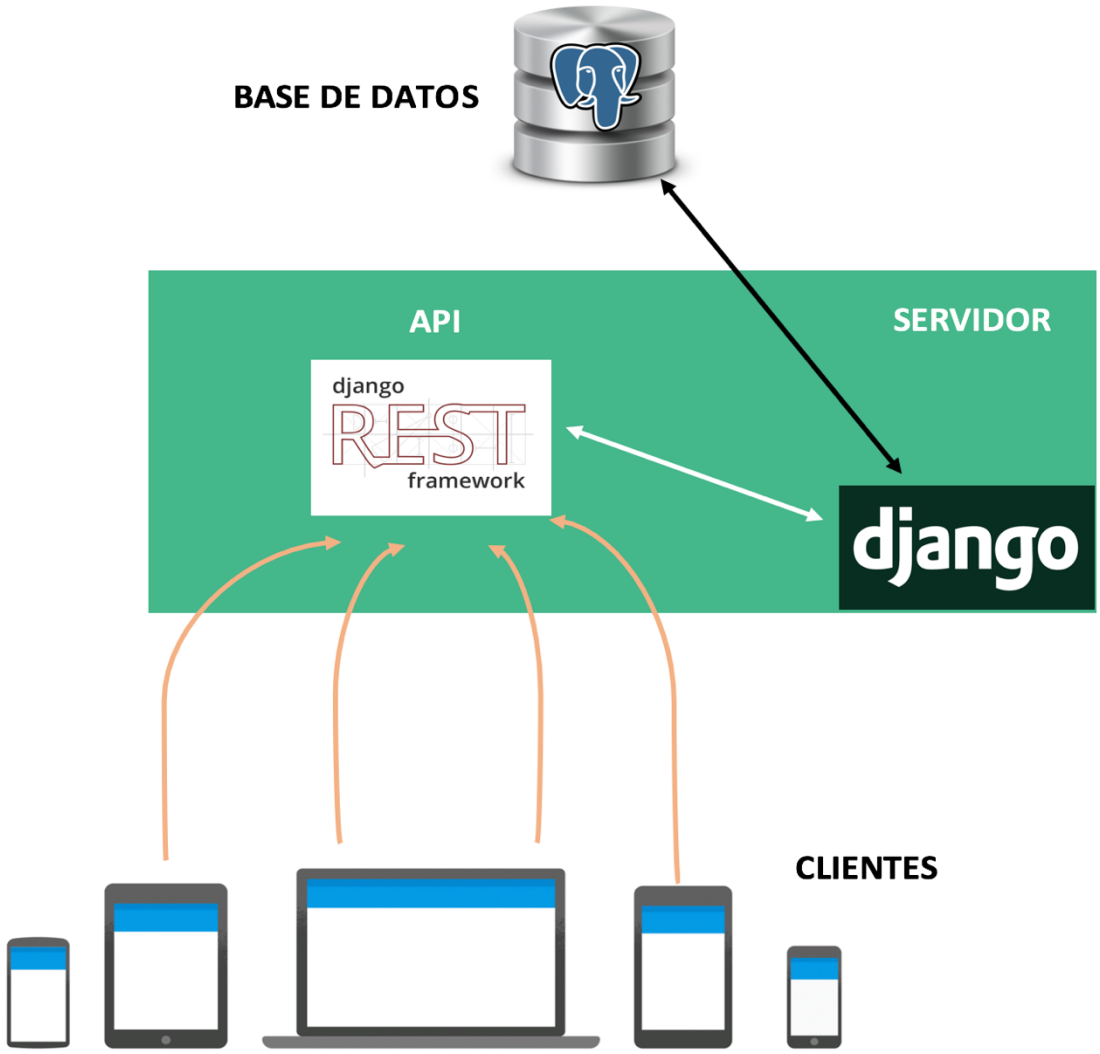

Figura 4.2. Arquitectura Cliente-Servidor

4.2.2 Arquitectura Modelo Vista Controlador

Al usar Django, hacemos uso del conocido patrón Modelo-Vista-Controlador, aunque desde el punto de vista de los creadores, quieran denominarlo con otro nombre. Es uno de los patrones más usados en el diseño de aplicaciones tanto webs como móviles, donde podemos diferenciar tres partes que ayudan a separar el funcionamiento interno del software y aislarlo de la información de la base de datos y la presentación de usuario.

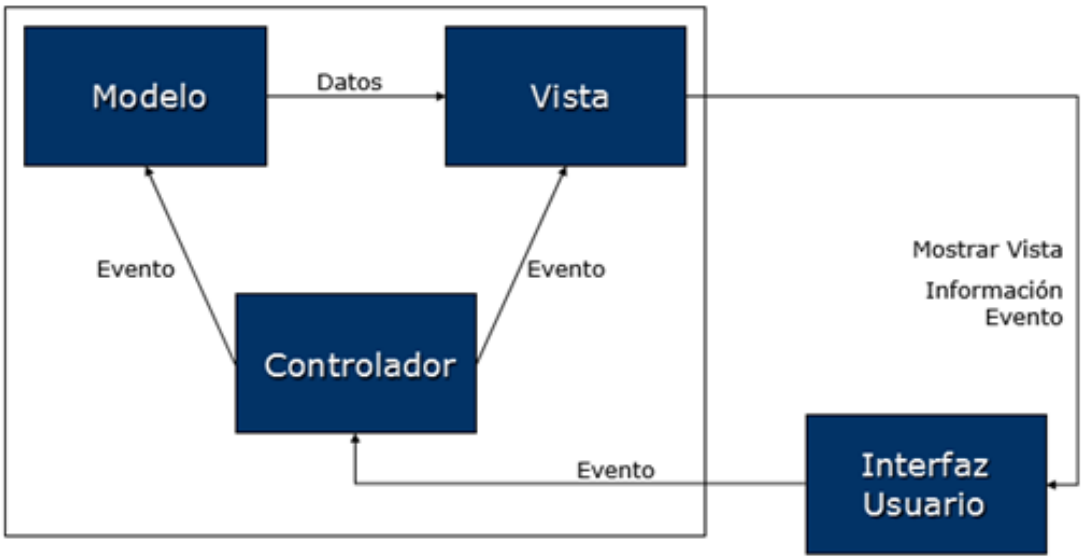

Figura 4.3. Arquitectura Modelo-Vista-Controlador

Modelo

Es la capa que trabaja con los datos, es decir, contiene los métodos para acceder a la información y también para actualizarlos. Estos datos estarán almacenados en el Sistema Gestor de Bases de Datos, PostgreSQL, y en nuestro caso los select, updates, inserts, etc. los realiza models.py de Django.

Vista

Es la capa que contiene el código de la aplicación que interactúa con los usuarios mediante las interfaces de usuario, es decir, en el caso de la aplicación web, es código HTML, CSS y JavaScript. Desde la vista se trabaja con los datos del modelo, pero no se realiza un acceso directo a ellos.

Controlador

Este es la capa entre la vista y el modelo que contiene el código que gestiona las respuestas a las acciones del cliente, como visualizar una página o algún tipo de información, etc.

4.3 Estructura y diseño

Hemos usado el framework bootstrap que nos permite usar muchos estilos de css y interacciones de JavaScript. Bootstrap facilita mucho la tarea de estructurar la aplicación, ya que tiene un sistema de que denominan de “rejilla” y nos permite organizar cada página en doce columnas y diferentes filas. Además, nos ofrece un gran conjunto de estilos ya aplicados, aunque también son totalmente personalizables. En nuestro caso se han personalizado algunos estilos como tener un formato como de folio en blanco por encima del fondo.

Los colores e iconos usados se han seleccionado usando entrevistas de usuario para validar ambos aspectos y comprobar que eran agradables y reconocibles por los usuarios.

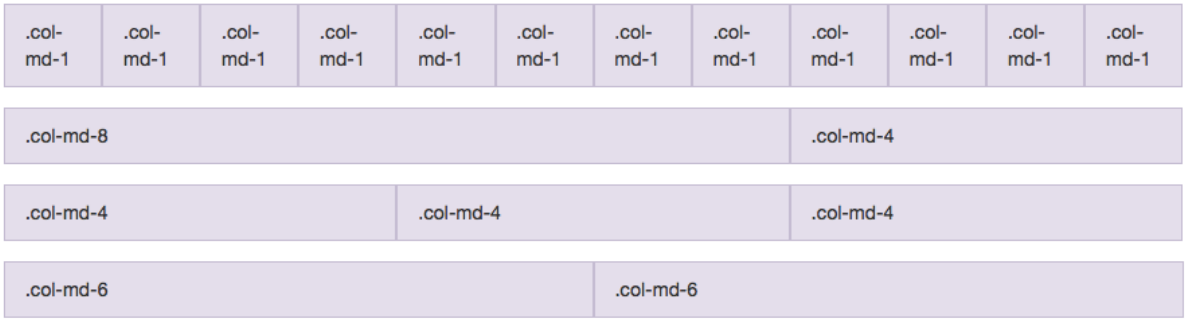

Figura 4.4. Sistema de "rejilla" de Bootstrap

4.4 Base de datos

Dentro de nuestra aplicación necesitamos una base de datos relacional que nos permita crear relaciones entre las tablas, ya que nuestro flujo de información así lo requiere. Además, realizamos un modelado de la base de datos multi-idioma, lo que hace que aumente un poco la dificultad de nuestra base de datos. Adicionalmente, nuestra aplicación requerirá obtener datos según las relaciones que ya se hayan creado entre diferentes entidades, tales como el usuario y una colección de películas o entre los usuarios entre sí.

Por esta razón, PostgreSQL encaja perfectamente con este perfil, ya que nos permite tener una base de datos relacional y funciona de forma adecuada con nuestro lenguaje de lógica de negocio en este caso Django.

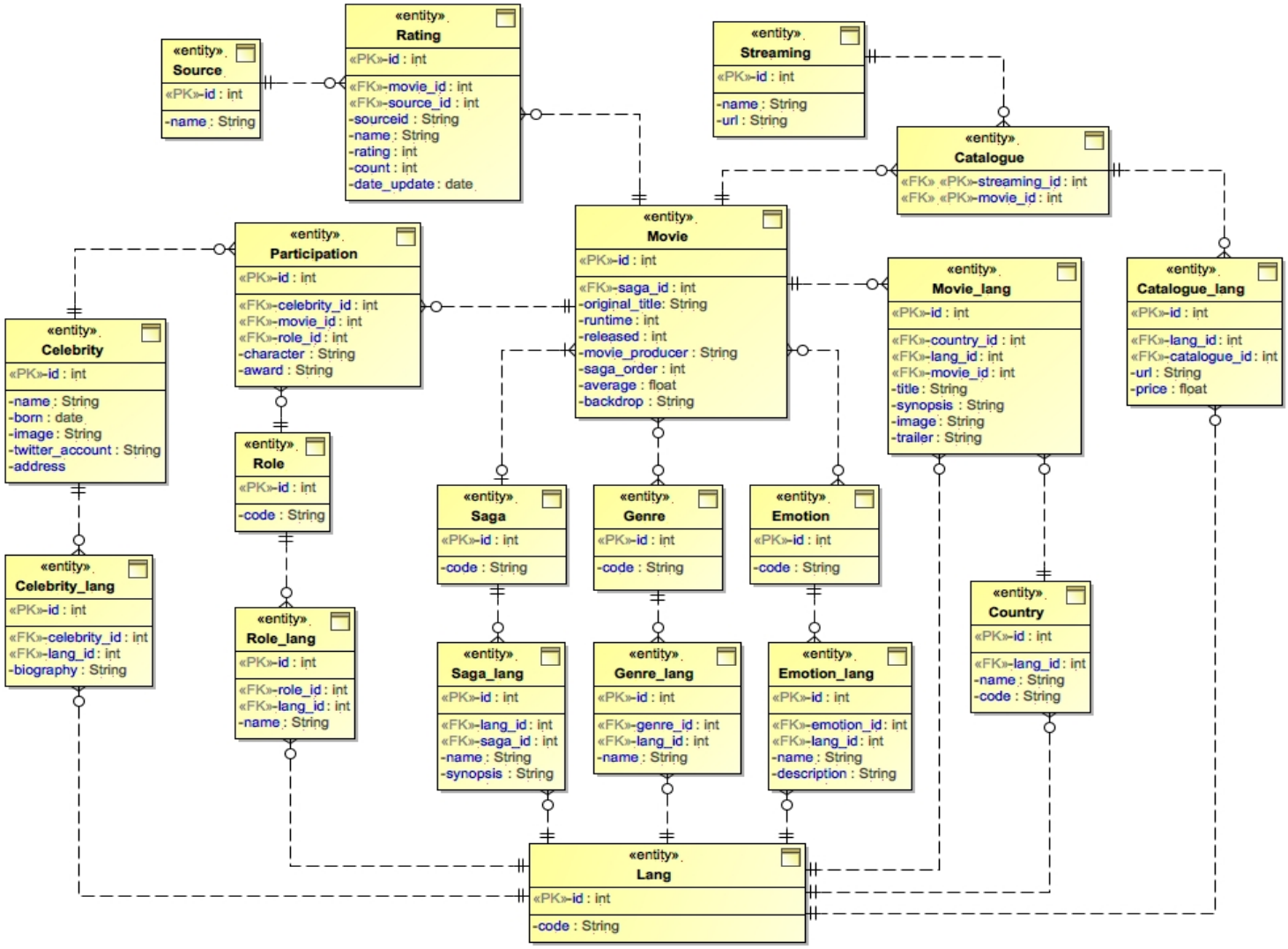

Figura 4.5. Esquema Base de datos de Películas

Como se puede ver en la imagen, cada relación que sea sensible de traducción a otros idiomas estará relacionada, a través de una relación muchos a muchos, con la tabla lang. De esta forma, de cada una de ellas habrá una tabla intermedia donde se encontrarán los datos en diferentes idiomas. También podemos observar que es Movie la que contiene multitud de relaciones con las otras tablas tales como celebridades, géneros, Sagas, Catálogos … En el caso especial de Movie con Celebridades tenemos una tabla intermedia llamada Participation, donde decimos qué rol tiene esa celebridad en nuestra película.

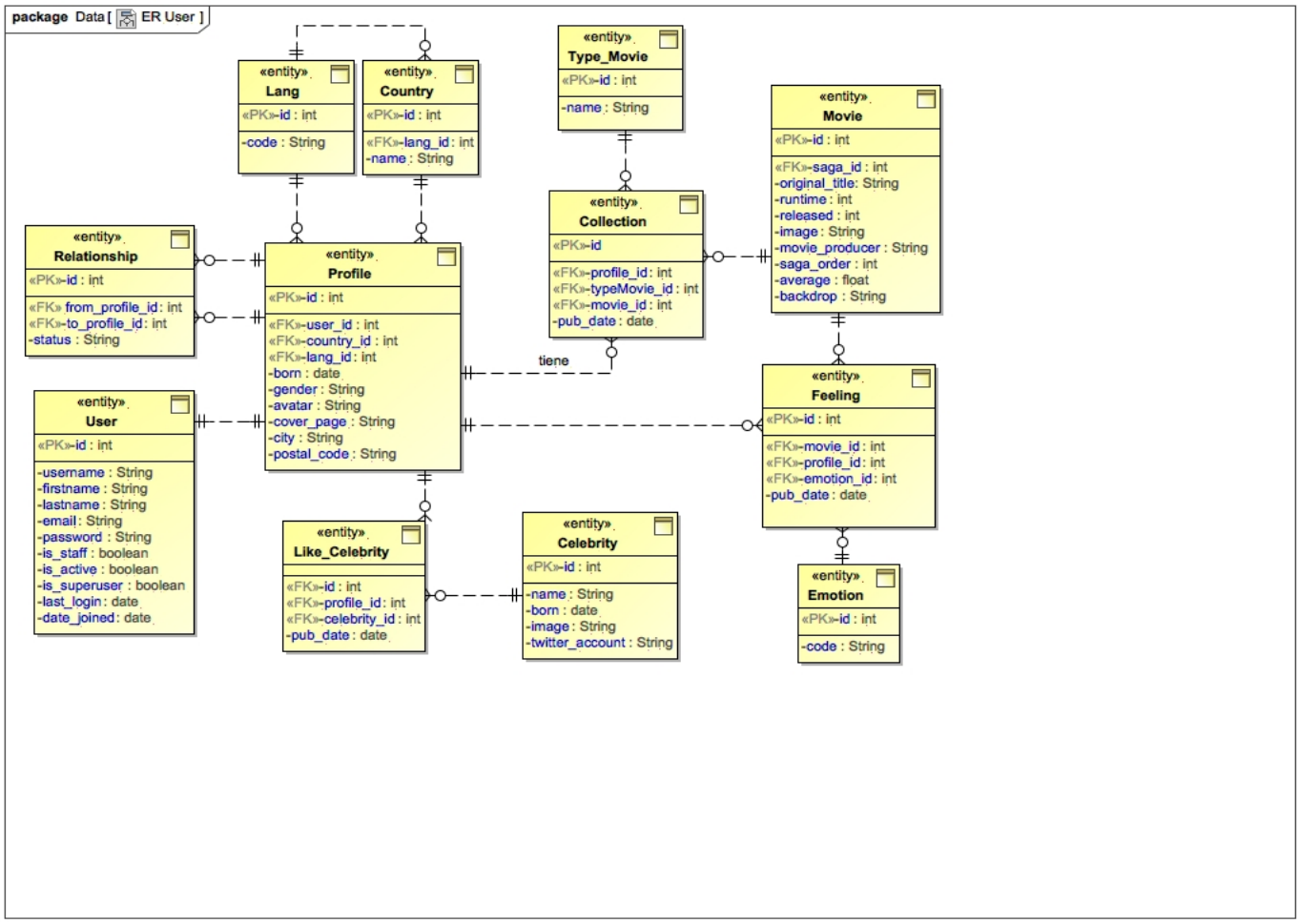

Figura 4.6. Esquema Base de datos de Usuario

En este segundo diagrama de entidad-relación mostramos las relaciones que tiene el usuario, donde hemos separado los datos de autenticación de los del perfil. Por ello hay dos tablas User y Profile con una relación uno a uno. Después, como podemos observar, es el Profile el que se relaciona con las demás tablas. Estas contienen información sobre el usuario como su colección de películas, las celebridades que le gustan y tendrán relaciones con otros usuarios. Esta última relación es asimétrica, ya que un usuario podrá seguir a otro, pero no viceversa. Asimismo, un usuario tendrá una relación con su idioma que nos permitirá relacionarlo con los registros que haya en su idioma en toda la base de datos.

Capítulo 5. Implementación e Instalación

5.2 Creación y configuración del proyecto

La creación y configuración del proyecto se podría separar en varias fases, todas ellas con ciertas instalaciones y configuraciones. Mostraremos cómo sería crear el proyecto desde cero y también cómo hacerlo a partir de GitHub o teniendo ya la carpeta del mismo. Todos estos procesos tiene una parte totalmente diferente y otra común.

En principio explicaremos los procesos previos que tienen en común las dos instalaciones. En primer lugar, tenemos que tener instalado pip, el gestor de paquetes de Python.

Una vez instalado pip procederemos a preparar nuestro entorno virtual (virtualenv), que nos permite aislar la configuración de los paquetes instalados de cada proyecto, por lo que podremos trabajar, por ejemplo, con diferentes versiones de Django o Python en proyectos diferentes.

Primero instalaremos virtualenv:

```
pip3 install virtualenv
```

Crearemos nuestro directorio donde crearemos nuestro virtualenv; en nuestro caso lo hemos llamado como el proyecto.

```
mkdir mooviest

cd mooviest

python3 –m venv venv

source venv/bin/activate
```

El último comando que mostramos se realiza para activar nuestro entorno virtual y ahora, una vez activado todo, lo que instalemos mediando pip3 lo estará de forma aislada en el entorno.

5.2.1 Creación del proyecto

Una vez montado todo lo explicado anteriormente podemos centrarnos en instalar y montar nuestro proyecto Django y añadirle todos los módulos o paquetes necesarios. En primer lugar instalaremos la versión de Django y crearemos nuestro proyecto.

```
pip install django==1.8

django-admin startproject mysite
```

El último comando creará los archivos y directorios con una estructura de proyecto Django y podremos ampliar instalando app o creando las nuestras propias.

En el siguiente paso crearemos nuestra propia aplicación que nos permitirá comenzar el proyecto.

```
python3 manage.py startapp movie
```

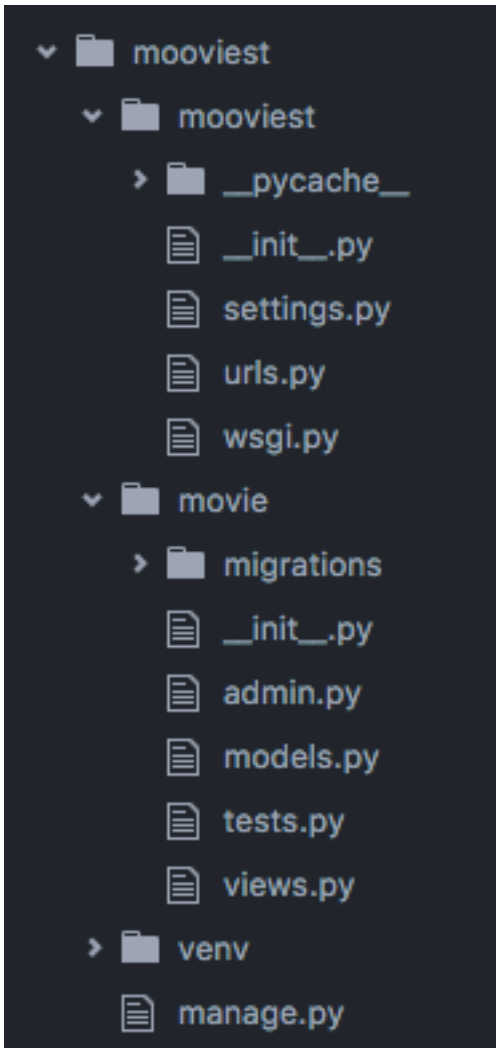

Figura 5.1. Estructura de carpetas en Django.

En Django se desarrolla creando aplicaciones que se empaquetan en su propio directorio. Como podemos ver en la <fig xxx>, tenemos una aplicación llamada movie. Además tenemos un subdirectorio llamado igual que el raíz desde donde se coordinan todas las aplicaciones mediante el urls.py y donde se configuran los ajustes del proyecto mediante el settings.py. Por esta razón, al crear una aplicación debemos añadirla al settings.py.

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'movie',
)
```

Figura 5.2. Código de aplicaciones instaladas.

Ahora procederemos a explicar cómo se organiza el directorio de una aplicación de django:

- admin.py : Se hace la configuración de los modelos en administrador.
- models.py: Se realiza la definición de las clases que serán nuestras entidades en la base de datos.
- Test.py: Se realizan los test unitarios para probar los controladores.
- views.py: Es el controlador donde cargaremos los templates.
- migrations: En esta carpeta se encuentra todas las migraciones que se realizan para crear los cambios en la base de datos.

Nosotros deberemos añadir a ese directorio dos nuevas carpetas que nos servirán en el desarrollo para crear los documentos html y sus estilos:

- templates: Es la carpeta donde django identifica que están todos los html de cara a cargalos en los views.py
- static: Se guardan los archivos estáticos de esta aplicación como imágenes, css o javascript.

Para instalar aplicaciones externas como Django Rest Framework se utiliza una vez más el gestor de paquetes pip y posteriormente añadiéndolo al settings.py:

```
pip3 install djangorestframework
```

5.2.2 Configuración del proyecto

En este apartado veremos cómo configurar nuestro proyecto a partir de tener el directorio raíz, ya sea por GitHub o descargándolo.

```
git clone https://github.com/JoseAntpr/mooviest.git
```

En este momento del proyecto, ya tendremos nuestra carpeta y nuestro entorno virtual creado como explicábamos anteriormente. Entonces tan solo debemos instalar las dependencias que ya existen en el proyecto en una carpeta llamada requirements.txt:

```
pip3 install –r requirements.txt
```

En el caso que quieras guardar nuevas dependencias en este documento, tan solo deberás utilizar otro comando para ello usando pip:

```
pip3 freeze > requirements.txt
```

5.2.3 Configuración de la base de datos

Para instalar la base de datos deberemos tener instalados PostgreSQL. Si no es así, deberemos instalarlo, bien desde la página del mismo, o a través del gestor de paquetes brew en MacOS o apt-get en Linux:

```
brew install postgresql
```

o

```
apt-get install postgresql
```

Una vez hecho esto crearemos nuestra base de datos y un usuario:

```
createdb mooviest
createuser –P
psql
>> GRANT ALL PRIVILEGES ON DATABASE mooviest TO root;
```

Si vemos la documentación de Django y como hemos comentado antes, la configuración de la base de datos se hace en el archivo settings.py. Una vez configurada, procederemos a ejecutar los comandos para crear los modelos y relaciones del proyecto.

```
DATABASES = {
    ‘default’:{
        ‘ENGINE’: ‘django.db.backends.postgreslq_psycopg2’,
        ‘NAME’: ‘mooviest’,
        ‘USER’: ’root’,
        ‘PASSWORD’: ’******’,
        ‘HOST’:’localhost’,
        ‘PORT’:’5432’,
    }
```

```
}


python3 manage.py makemigrations
python3 manage.py migrate
```

5.2.4 GitHub

Como hemos comentado anteriormente, usaremos GitHub para mantener un control de versiones y un correcto entorno de desarrollo y flujo de trabajo, Por ello hemos creado un repositorio privado en GitHub donde tener alojado el código y poder hacer uso del VCS (*Version Control System*) git. Este sistema nos permite volver a versiones anteriores o revisar los últimos cambios realizados, permitiendo una mejor coordinación de un equipo de desarrollo.

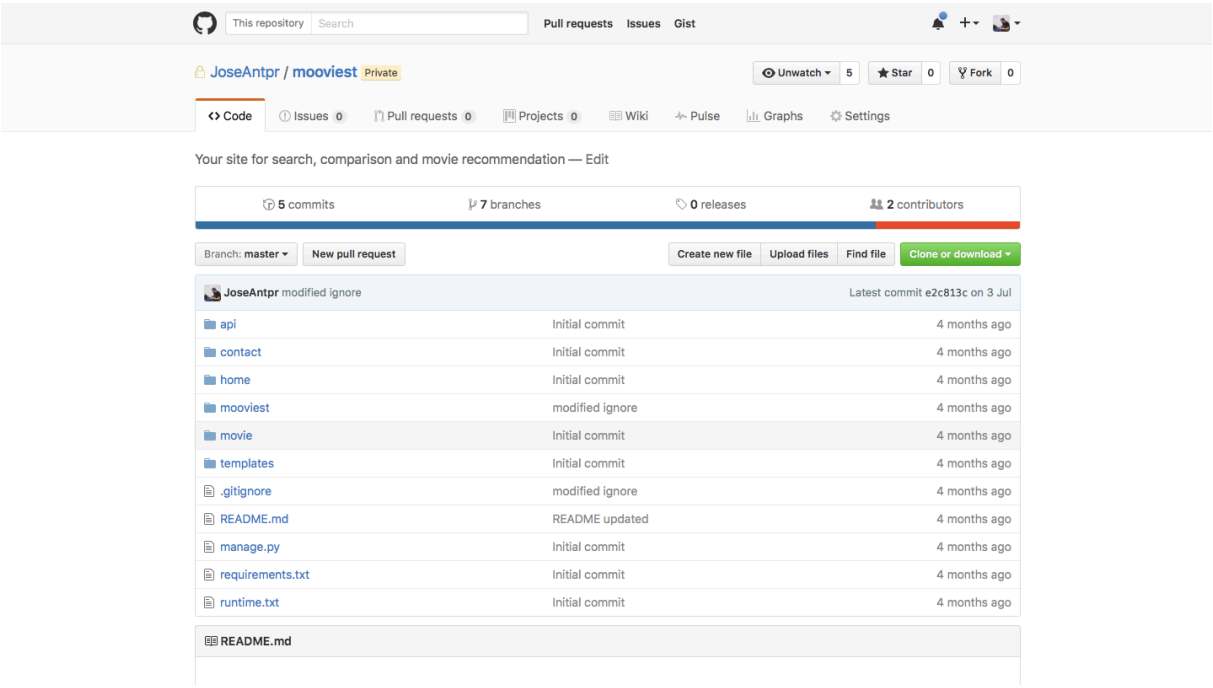

Figura 5.3. Repositorio de GitHub.

5.3 Aplicación Web

5.3.1Página de inicio

La página principal de la aplicación muestra diferentes secciones de las últimas películas clasificadas por los usuarios. Además, muestra el menú en el que se puede ver la caja de texto donde se realiza la búsqueda de nuevas películas o usuarios. Por otra parte, los enlaces para iniciar sesión o registrarte, si no se ha hecho todavía; en el caso contrario, mostraría una imagen del perfil de usuario y un menú desplegable donde podrá navegar a su propio perfil, ajustes o cerrar sesión.

En primer lugar, podríamos introducir la película que queremos buscar; podríamos introducir alguna palabra que contenga tanto en su título original como en el del idioma que tengamos en nuestra cuenta. Para el caso de buscar un usuario sería igual introducir una cadena de texto que contenga bien su nombre de usuario o su nombre o apellidos.

El código HTML de este template se encuentra en home/templates/home/index.html y su correspondiente views, encargada de manejar los datos introducidos y cargar el template correspondiente en home/views.py

Al pulsar en el botón de búsqueda, obtendremos los resultados en una página de detalle de la búsqueda que mostraremos en el siguiente apartado. En el caso de que solo encuentre un objeto, ya sea una película o un usuario, entonces mostrará bien el perfil de usuario o la página de detalle de esa película.

5.3.2 Página de búsqueda

La página de detalle de la búsqueda se encuentra en la ruta movie/templates/movie/search.html y es llamada por el controlador siempre que se pulsa en el botón de búsqueda, que se encuentra en el menú que es siempre visible en la parte superior de la aplicación. Esta página muestra en orden las películas y usuarios encontrados según los caracteres introducidos en la caja de texto, así como información relevante de cada uno de los elementos buscados. En el caso de las películas, muestra la carátula y la sinopsis. En el caso del usuario, su imagen de perfil y su nombre de usuario. En esta pantalla puedes seleccionar cualquier elemento de los encontrados o navegar a cualquiera de las posibilidades que ofrecen el menú superior o el inferior.

5.3.3 Inicio de sesión o Registro

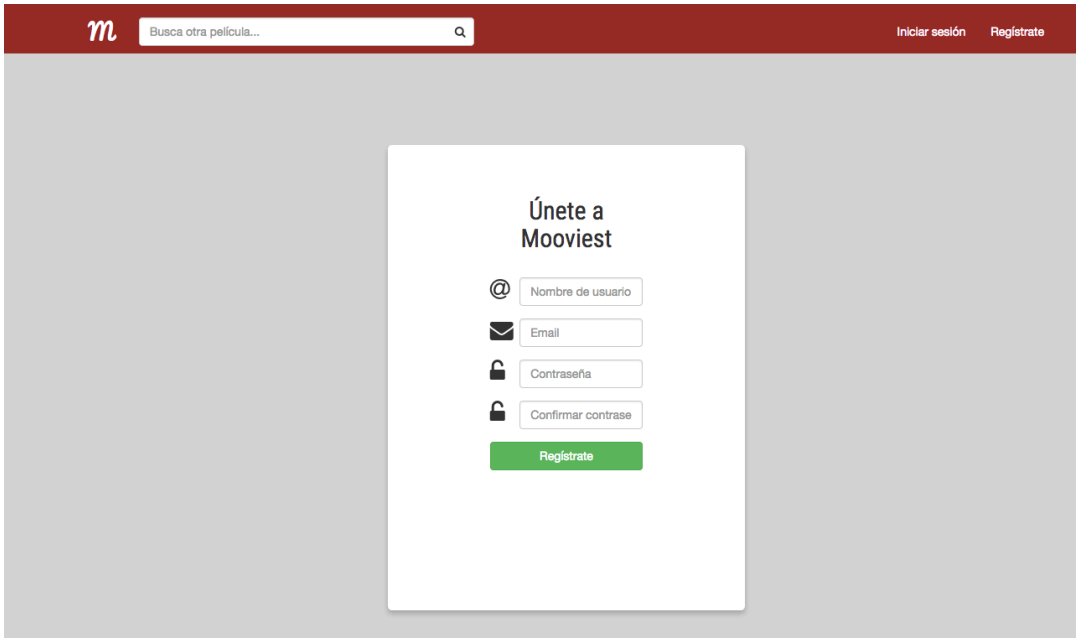

Figura 5.4 Página de Registro.

En el caso del Inicio de sesión o el registro, mostramos dos formularios donde nos pedirán unos datos de entrada que nos permitirán ambos acceder igualmente a la aplicación.

En primer lugar, explicaremos el formulario de registro. En este caso, debemos introducir todos los datos obligatoriamente para poder crear una cuenta para acceder a la aplicación. Recalcar que, en el momento del registro, en el controlador <ruta>, se obtiene también un dato más. Más concretamente, el idioma del navegador del usuario y, posteriormente, le asignamos ese idioma a su cuenta.

```
user_model =
User.objects.create_user(username=username,password=password,email=email)


    user_profile = Profile()

    user_profile.user = user_model

    user_profile.lang = Lang.objects.get(code=lang)


    user_profile.save()
```

Para el caso del inicio de sesión, podremos acceder a nuestro perfil tan solo introduciendo nombre de usuario o correo electrónico, además de la contraseña. Si todo ha ido bien, obtendremos el acceso a la aplicación.

5.3.4 Perfil y ajustes de usuario.

En esta vista tendremos la opción de visualizar el perfil. Esta vista está estructurada en dos grandes partes:

Cabecera: Donde observamos una imagen de fondo, la foto del perfil del usuario e información de usuario como el nombre de usuario. También podremos ver un botón que, si el perfil es el nuestro propio, nos indicará que podemos editarlo en los ajustes, mientras que en otro caso, nos permite seguir o dejar de seguir al usuario del perfil en cuestión.

Cuerpo: Observaremos un menú inicial que nos permite ver las diferentes listas de películas del usuario, sus seguidores o las personas a las que sigue. En la vista de listas habrá un submenú que nos dejará ver en detalle todas las películas que el usuario ha clasificado.

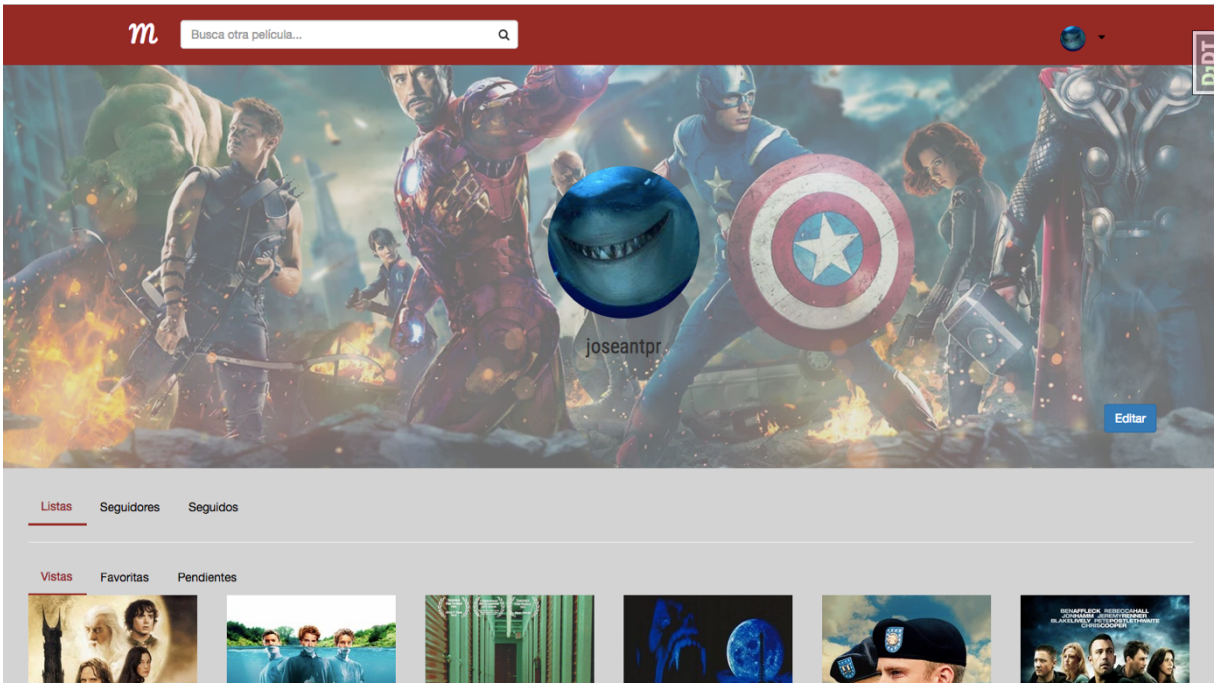

Figura 5.5. Perfil de un usuario.

También referente al usuario podemos editar los datos introducidos en el registro, además de otros campos como son su imagen de perfil, nombre, apellidos, fecha de nacimiento, género, ciudad, código postal o cambiar la contraseña.

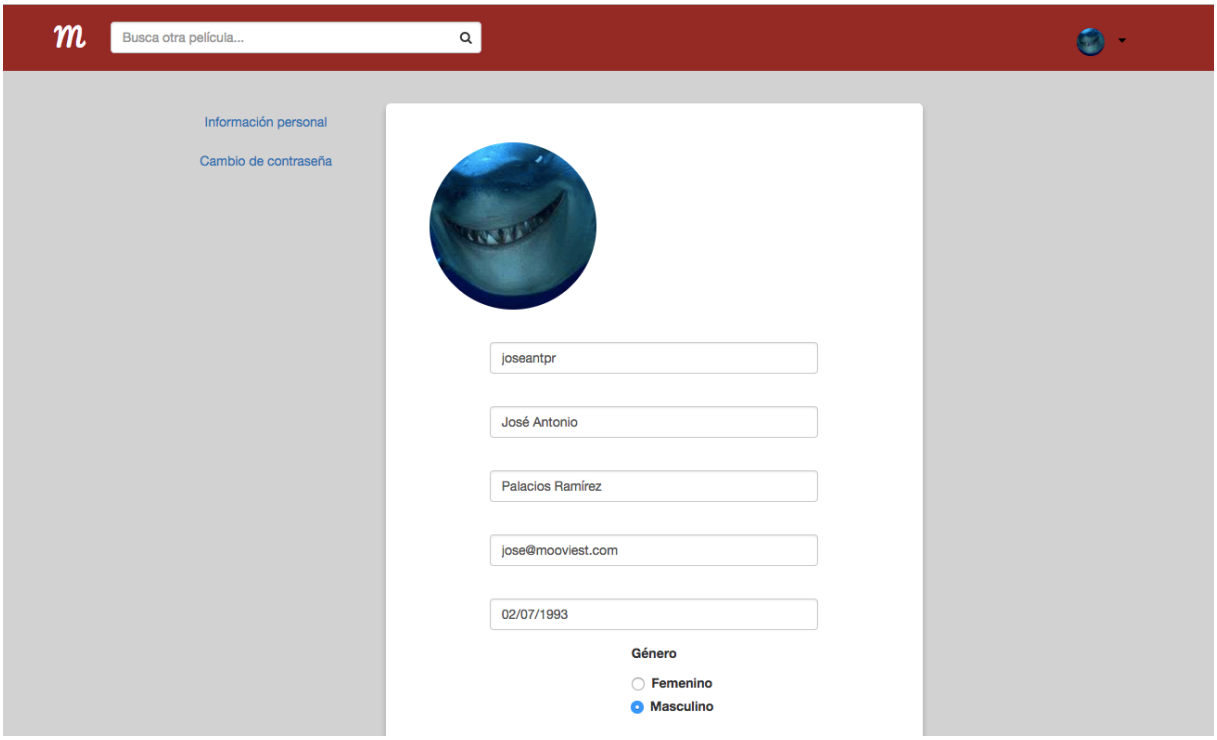

Figura 5.6. Ajustes de usuario

Para actualizar el perfil correctamente se validan algunos campos; en este caso nombre de usuario y email que deben ser únicos, este tipo de validaciones se realizan en validadores que se crean en los formularios de Django.

```
def clean_username(self):
        #Comprueba que no exista un username igual en la db
        username = self.cleaned_data['username']
        if not self.user.username == username and User.objects.filter(username=username.lower()) :
            raise forms.ValidationError('Nombre de usuario ya registrado.')
        else:
            return username.lower()
    def clean_email(self):
        email = self.cleaned_data['email']
        if not self.user.email == email:
            raise forms.ValidationError('Ya existe un email igual en la db')
        return email
```

5.3.5 Detalle de una película

Este template corresponde a movie/templates/movie/index.html y en ella mostramos la información detallada de una película con su carátula, sinopsis, título, duración, reparto, puntuaciones, etc. Sobre las puntuaciones, decir que esta información es proporcionada por otros sitios web como IMDb, Tviso o FilmAffinity. Además, también se incluyen los botones característicos para clasificar las películas.

Figura 5.7. Página detalle de una película.

Los botones son los que se indica en la Figura 5.7 donde, si en algún momento, alguno está seleccionado, perderá su color característico a saber:

- Rojo: Muestra un icono de una cruz y significa que la película no me interesa.
- Azúl: Con un icono marca páginas que permite clasificar la película como pendiente.
- Verde: Icono con un ojo y nos indica que la hemos visto.
- Amarillo: Icono con una estrella que nos indica que esa película es de nuestras favoritas.

Para clasificar las películas llamaríamos al view changeCollection para actualizar su estado.

5.4 API REST

En esta sección se explica las partes más importantes del desarrollo de la API, así como la documentación de las peticiones construidas.

5.4.1 Desarrollo

Autenticación

Comenzaremos explicando el sistema de autenticación de nuestra aplicación basado en tokens. Este token es único y se le asigna a un usuario. Este sistema es perfecto para uso de aplicaciones de escritorio o móviles.

```
from rest_framework.authtoken.models import Token

from rest_framework.authentication import TokenAuthentication


token = Token.objects.get_or_create(user=user)[0].key
```

Permisos

Los permisos determinan qué peticiones deben ser aceptadas o denegadas; se declaran en los viewsets que son los encargados de recibir las peticiones de la API.

En Django Rest Framework hay muchos tipos de permisos predeterminados como pueden ser:

- AllowAny: no restringe el acceso a nadie.
- IsAuthenticated: este permiso deniega el acceso a todos aquellos que no estén autenticados.
- IsAdminUser: este último restringe el acceso a todos aquellos que no sean parte administradores.
- Etc.

Se declararía así en los viewsets.

```
permission_classes = (IsAuthenticated,)
```

También podemos crear nuestros permisos personalizados, permitiendo el máximo control sobre los métodos.

```
from rest_framework.permissions import BasePermission

class UserPermission(BasePermission):

    def has_permission(self, request, view):
```

```
        if view.action == 'create' or view.action == 'login' or view.action in ['retrieve', 'update', 'destroy']:
            return True
        elif view.action == 'list':
            return False
        else:
            return request.user.is_authenticated()



    def has_object_permission(self, request, view, obj):
if view.action == 'retrieve':
                    return request.user.is_authenticated()
            else:
                    return request.user.is_superuser or request.user == obj
```

Todo permiso personalizado tiene dos métodos: has_permissions y has_object_permission. En el primero se comprueba si el usuario puede realizar cualquiera de las peticiones. Si su valor es True, entonces en el segundo método es donde se evalúa el permiso para un objeto. En nuestro caso evalúa si el usuario puede ver su perfil.

Serializers

Los serializadores permiten convertir datos complejos como consultas o instancias del modelo de Python a JSON, XML u otros tipos de contenido. También permiten la deserialización y validar los datos.

Los serializadores de Django Rest Framework tienen muchos campos predefinidos que nos permiten validar automáticamente ciertos tipos de datos, por ejemplo, que el tipo sea de correo electrónico, número de caracteres o si es el campo es requerido o no.

```
class UserSerializer(serializers.ModelSerializer):
```

```
    profile = ProfileSerializer(partial=True)

    class Meta:
        model = User
        fields = ('id', 'username', 'first_name', 'last_name', 'email', 'profile')




    def update(self, instance, validated_data):
        profile_data = validated_data.pop('profile')

        profile = instance.profile

        instance.username = validated_data.get('username', instance.username)
        instance.first_name = validated_data.get('first_name', instance.first_name)
        instance.last_name = validated_data.get('last_name', instance.last_name)
        instance.email = validated_data.get('email', instance.email)
        instance.save()

        profile.born = profile_data.get('born', profile.born)
        profile.gender = profile_data.get('gender', profile.gender)
        profile.avatar = profile_data.get('avatar', profile.avatar)
        profile.city = profile_data.get('city', profile.city)
        profile.postalCode = profile_data.get('postalCode', profile.postalCode)
        profile.save()

        return instance
```

Podemos observar que en este caso los tipos de los campos los cogerá del modelo de Django, pudiendo ser CharField, IntegerField en el caso del id y EmailField, en el

caso del correo. Los serializers también crean y guardan las instancias con los métodos create() y update().

Como decíamos, los serializers también pueden validar los datos según parámetros, pero hay casos donde es necesario hacer una validación más compleja, por lo que podemos redefinir el validador del campo.

```
def validate_username(self, data):
    user = User.objects.filter(username=data.lower())
    if self.instance is not None and self.instance.username != data and User.objects.filter(username=data.lower()):
        raise serializers.ValidationError('Nombre de usuario ya registrado.')
    else:
        return data
```

En este caso validamos el campo de nombre de usuario donde comprobamos si ya ha sido registrado anteriormente.

Viewsets

Los viewsets gestionan las peticiones y para ello tiene manejadores como list(), create(), retrieve(), update(), partial_update o destroy(). Son los encargados de devolver los datos en el formato.

```
class UserViewSet(GenericViewSet):
    authentication_classes = (TokenAuthentication,)
    permission_classes = (UserPermission,)

    def retrieve(self,request,pk):
        user = User.objects.get(pk=pk)
        profile = user.profile
        lang = Lang.objects.get(pk = profile.lang.id)

        return Response(
            {
                'user':{
```

```
                'id': user.id,
                'username': user.username,
                'first_name': user.first_name,
                'last_name': user.last_name,
                'email': user.email,
                'profile':{
                    'born': profile.born,
                    'avatar': profile.avatar.url,
                    'city': profile.city,
                    'gender': profile.gender,
                    'postalCode': profile.postalCode,
                    'lang': {
                        'code': lang.code
                    },
                },
            },
            'status':status.HTTP_200_OK,
        }
    )

def update(self,request,pk=None):
    data = request.data
    http_code = ""
    errors = None

    user = get_object_or_404(User,pk=pk)
    serializer = UserSerializer(instance=user,data=data)

    if serializer.is_valid():
```

```
            user = serializer.save()
            data =  serializer.data
            http_code = status.HTTP_200_OK

        else:
            data = None
            http_code = status.HTTP_400_BAD_REQUEST
            errors = serializer.errors

        return Response(
            {
                'user': data,
                'status': http_code,
                'errors': errors
            }
        )
```

En este caso podemos observar el método retrieve que tan solo hace una consulta del usuario y lo devuelve en formato JSON, y el método update que llamará al método del serializer para guardar los datos recibidos una vez hayan sido validados.

5.4.2 Peticiones

Explicaremos las llamadas a la API de nuestro proyecto que hemos necesitado para que las aplicaciones se puedan comunicarse con nuestro servidor y, en concreto, con la base de datos. Primero detallaremos las peticiones relacionadas con un usuario, para luego pasar con las relacionadas con una película.

User

- **AUTH - PostLogin:** Iniciar sesión en el sistema.

  (POST) /users/login/

  Datos del formulario para x-www-form-urlencoded

Campo	Tipo	Descripción

username	*Text*	Nombre de usuario o email registrado en el sistema
password	*Text*	Contraseña asociada al usuario o email introducido

Respuesta


```
{
  "message":"Login successfully",
  "status":200,
  "user":{
    "username":"movie",
    "profile":{
      "avatar":"/media/user/default/no-image.png",
      "lang":{
        "code":"es"
      }
    },
    "id":9,
    "email":"movie@mooviest.com"
  },
  "token":"8a0ac632536d7d1ac5db90f6f05338cef2778516"
}
```


Obtenemos los datos del usuario y su perfil, además del *token*, necesario para realizar el resto de peticiones que no son de tipo *AUTH* (Autorizadas).

Error 4xx


```
{
  "message": "User or password incorrect",
  "status": 404,
  "user": null,
```

```
  "token": null
}
```


Si no existe un usuario en el sistema con los campos introducidos nos devolverá un mensaje de error.

- **AUTH - PostSignUp:** Registrar un usuario en el sistema.
  (POST) /users/

  Datos del formulario para x-www-form-urlencoded

Campo	Tipo	Descripción
username	*Text*	Nombre de usuario
email	*Text*	Email
password	*Text*	Contraseña
profile.lang.code	*Text*	Código de idioma. Actualmente los valores posibles son: *es*: Español *en*: Inglés

  Respuesta


```
{
  "status": 201,
  "user": {
    "id": 9,
    "username": "movie",
    "email": "movie@mooviest.com",
    "password":
"pbkdf2_sha256$20000$wiNMenJqVebu$DUTwrahkp1n+teQZbnhGrcWI+e+8
toKN6hC20xz5QgQ=",
    "profile": {
      "lang": {
        "code": "es"
      },
      "avatar": "/media/user/default/no-image.png"
    }
  },
  "errors": null,
  "token": "8a0ac632536d7d1ac5db90f6f05338cef2778516"
```

}

Obtenemos los datos del usuario y su perfil, además del *token*, necesario para realizar el resto de peticiones que no son de tipo *AUTH* (Autorizadas).

Error


```
{
  "status": 400,
  "user": null,
  "errors": {
    "username": [
      "Ya existe un usuario con ese nombre de usuario."
    ],
    "email": [
      "Ya existe un usuario con ese email."
    ]
  },
  "token": null
}
```


Si ya existe un usuario en el sistema con los campos introducidos, nos devolverá un campo de error con los mensajes correspondientes.

Nota: Para el resto de peticiones se utiliza un método de autenticación por Token, para dar seguridad a la API y que solo los usuarios registrados puedan realizar peticiones al resto de métodos. Se añadirá un campo de *Authorization* a la cabecera como el siguiente.

Authorization: Token {token}

Campo	Tipo	Descripción
token	*Text*	Token asociado a usuario del sistema

- **User - GetUserProfile:** Obtener el perfil de un usuario.
  (GET) /users/{id}/

  URL

Campo	Tipo	Descripción

id	*Number*	Id de un usuario registrado en el sistema

Respuesta


```
{
  "status": 200,
  "user": {
    "first_name": "",
    "last_name": "",
    "username": "movie",
    "profile": {
      "born": null,
      "gender": null,
      "avatar": "/media/user/default/no-image.png",
      "lang": {
        "code": "es"
      },
      "postalCode": null,
      "city": null
    },
    "id": 9,
    "email": "movie@mooviest.com"
  }
}
```


Obtenemos todos los datos del usuario y su perfil.

- **User - UpdateUserProfile:** Actualizar el perfil de un usuario.
  (PUT) /users/{id}/

  URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema

Datos del formulario para multipart/form-data

Campo	Tipo	Descripción
username	*Text*	Nombre de usuario
profile.lang.code	*Text*	Código de idioma

first_name (opcional)	*Text*	Nombre
last_name (opcional)	*Text*	Apellidos
email (opcional)	*Text*	Email
profile.born (opcional)	*Date*	Fecha de nacimiento
profile.city (opcional)	*Text*	Ciudad
profile.postalCode (opcional)	*Text*	Código postal
image (opcional)	*Image*	Imagen del perfil

Respuesta

Devolvería la misma respuesta que en la petición **GetUserProfile.**

- **User - GetSwipeList:** Obtener la lista de películas para el sistema de clasificación (*swipe*), es decir, películas que no están clasificadas en ninguna lista del usuario.
  (GET) /users/{id}/swipelist/

  URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema

Respuesta


```
{
  "count": 10,
  "next": null,
  "previous": null,
  "results": [
      {
    "movie_lang_id": 54625,
    "image": "EXTERNAL#pelis/HP9YTUHXX6",
    "title": "Cartas desde la locura",
    "collection": null,
    "id": 28689,
    "average": 0
  },
      {
```

```
    "movie_lang_id": 48254,
    "image": "EXTERNAL#pelis/VHRDRXK7X4",
    "title": "Ring of fire II: Sangre y acero",
    "collection": null,
    "id": 25274,
    "average": 0
   },
   {
    "movie_lang_id": 7581,
    "image": "/14/f3/14f30376096122243ff4fe3de8db0cb6.jpg",
    "title": "Uno de los nuestros",
    "collection": null,
    "id": 3813,
    "average": 0
   },
                    .
                    .
                    .
  ]
}
```


Obtenemos la información necesaria de 10 películas para mostrar en el sistema de clasificación (*swipe*). Nos devuelve su título en el idioma del perfil del usuario, la carátula (*cover*) de la película, la puntuación de nuestro sistema, su id asociada a nuestra base de datos y su tipo de colección que, en este caso, al ser para el sistema de clasificación y no estar aún clasificada nos devolverá *null*.

- **User - GetCollectionList:** Obtener la lista de películas de un tipo concreto.
  (GET) /users/{id}/collection/?name={name}&page={page}

  URL

Campo	Tipo	Descripción
id	*Number*	Id de un usuario registrado en el sistema
name	*Text*	Nombre de la colección. Los tipos de colección son: seen, favourite, watchlist, blacklist
page	*Number*	Número de página

Respuesta


```
{
  "count": 174,
  "next": "/users/9/collection/?name=favourite&page=2",
  "previous": null,
  "results": [
    {
      "movie_lang_id": 2919,
      "image": "/a4/9d/a49dd25d5df38f2e97c4bfd7d4875684.jpg",
      "title": "El Señor de los Anillos: La Comunidad del Anillo",
      "collection": {
        "typeMovie": "favourite",
        "id": 1
      },
      "id": 1468,
      "average": 0
    },
        {
      "movie_lang_id": 6140,
      "image": "/1d/8f/1d8f6066a9ad1cb91211bcbfca5915f2.jpg",
      "title": "Piratas del Caribe: La maldición de la Perla Negra",
      "collection": {
        "typeMovie": "favourite",
        "id": 9
      },
      "id": 3088,
      "average": 0
    },
        {
      "movie_lang_id": 6136,
      "image": "/1c/3e/1c3e15050ceaf0e4edb15cdfb3c35867.jpg",
      "title": "Piratas del Caribe: El cofre del hombre muerto",
      "collection": {
        "typeMovie": "favourite",
        "id": 13
      },
      "id": 3086,
      "average": 0
    }
  ]
}
```

Obtenemos un resultado paginado con la información necesaria, para poder mostrar una previsualización de las películas. Nos devuelve su título en el idioma del perfil del usuario, la carátula (*cover*) de la película, la puntuación de nuestro sistema, su id asociada a nuestra base de datos y su tipo de colección.

- **AUTH - PostMovieCollection:** Clasificar una película en la colección de un usuario determinada.
  (POST) /collection/

  Datos del formulario para x-www-form-urlencoded

Campo	Tipo	Descripción
user	*Number*	Id de un usuario del sistema
movie	*Number*	Id de una película del sistema
typeMovie	*Text*	Nombre de la colección. Los tipos de colección son: seen, favourite, watchlist, blacklist

  Respuesta


```
{
  "id": 854,
  "user": 9,
  "movie": 3088,
  "typeMovie": "favourite",
  "pub_date": "2016-11-06T07:44:48.756881Z"
}
```


  Obtenemos el resultado de haber clasificado una película en una colección determinada, los id de la tabla ***Collection***, del usuario, de la película, el tipo de la colección y la fecha de clasificación.

- **AUTH - PatchMovieCollection:** Actualizar la colección de una película de un usuario.
  (PATCH) /collection/{id}/

  URL

Campo	Tipo	Descripción

id	*Number*	Id de la tabla ***Collection*** donde está clasificada la película que queremos actualizar

Datos del formulario para x-www-form-urlencoded

Campo	Tipo	Descripción
typeMovie	*Text*	Nombre de la colección. Los tipos de colección son: seen, favourite, watchlist, blacklist

Respuesta


```
{
  "id": 854,
  "user": 9,
  "movie": 3088,
  "typeMovie": "seen",
  "pub_date": "2016-11-06T07:48:47.107361Z"
}
```


Obtenemos el resultado de haber actualizado el tipo de colección de una película, los id de la tabla ***Collection***, del usuario, de la película, el tipo de la colección y la fecha de clasificación.

Movie

- **Movie - GetMovieDetail:** Obtener la información detallada de una película.
  (GET) /movie/{id}/?movie_lang_id={movie_lang_id}&user_id={user_id}

  URL

Campo	Tipo	Descripción
id	*Number*	Id de la película a obtener
movie_lang_id	*Number*	Id de la tabla ***Movie_lang*** que identifica a la película en un idioma.
user_id	*Number*	Id del usuario que obtiene la película.

Respuesta


```
{
  "title": "Piratas del Caribe: La maldición de la Perla Negra",
  "participations": [
    {
      "celebrity": {
        "id": 6687,
        "name": "Terry Rossio",
        "born": null,
        "image": "/0c/c8/0cc8bb88f9135835849485bf8f27cf7b.jpg",
        "twitter_account": "",
        "address": ""
      },
      "role": "Escritor",
      "character": "",
      "award": ""
    },
        {
      "celebrity": {
        "id": 11741,
        "name": "Gore Verbinski",
        "born": null,
        "image": "/47/3b/473b2276bf3a902a427642afa76a87f8.jpg",
        "twitter_account": "",
        "address": ""
      },
      "role": "Director",
      "character": "",
      "award": ""
    },
    {
      "celebrity": {
        "id": 9390,
        "name": "Keira Knightley",
        "born": null,
        "image": "/ec/cf/eccf02146ae943544067b273437c487b.jpg",
        "twitter_account": "",
        "address": ""
      },
      "role": "Actor",
      "character": "Elizabeth Swann",
      "award": ""
```

```
    },
    {
      "celebrity": {
        "id": 12913,
        "name": "Orlando Bloom",
        "born": null,
        "image": "/f2/ce/f2cea9813f0850ef86cdd64660849247.jpg",
        "twitter_account": "",
        "address": ""
      },
      "role": "Actor",
      "character": "Will Turner",
      "award": ""
    },
        {
      "celebrity": {
        "id": 1020,
        "name": "Johnny Depp",
        "born": null,
        "image": "/f0/2c/f02cf77dbedcf318cedf4e436ebfbcb6.jpg",
        "twitter_account": "",
        "address": ""
      },
      "role": "Actor",
      "character": "Jack Sparrow",
      "award": ""
    }
                              .
                              .
                              .
  ],
  "collection": {
    "typeMovie": "seen",
    "id": 854
  },
  "movie_lang_id": 6140,
  "movie_producer": "Walt Disney Pictures | Jerry Bruckheimer Films",
  "synopsis": "El aventurero Capitán Jack Sparrow recorre las aguas caribeñas. Pero sus andanzas terminan cuando su enemigo, el Capitán Barbossa le roba su barco, el Perla Negra, y ataca la ciudad de Port Royal, secuestrando a Elizabeth Swann, hija del Gobernador. Will Turner, el amigo de la infancia de Elizabeth, se une a Jack para rescatarla y recuperar el Perla Negra. Pero el prometido de Elizabeth, Comodoro Norrington, les persigue a bordo del HMS Dauntless. Además, Barbossa y su tripulación son víctimas de
```

```
un conjuro por el que están condenados a vivir eternamente, y a transformarse cada noche en esqueletos vivientes, en fantasmas guerreros.",
  "id": 3088,
  "genres": [
    {
      "name": "Fantasía"
    },
    {
      "name": "Acción"
    },
    {
      "name": "Aventura"
    }
  ],
  "runtime": 143,
  "original_title": "Pirates of the Caribbean: The Curse of the Black Pearl",
  "released": 2003,
  "average": 0,
  "country": null,
  "backdrop": "/1d/8f/1d8f6066a9ad1cb91211bcbfca5915f2.jpg",
  "ratings": [
    {
      "name": "IMDb",
      "rating": 81,
      "count": 811554,
      "date_update": "2016-08-24"
    },
    {
      "name": "Tviso",
      "rating": 86,
      "count": 70896,
      "date_update": "2016-08-24"
    }
  ],
  "image": "/1d/8f/1d8f6066a9ad1cb91211bcbfca5915f2.jpg"
}
```


Obtenemos la información detallada de una película.

- **Movie - GetSearchMovies:** Obtener el resultado de buscar películas por su título original o por el del idioma del usuario que busca las películas.
  (GET) /movie_lang/?title={title}&code={code}&page={page}

URL

Campo	Tipo	Descripción
title	*Text*	Título de una película a buscar.
code	*Text*	Código de idioma. Actualmente los valores posibles son: *es*: Español *en*: Inglés
page	*Text*	Número de página

Respuesta


```
{
  "count": 6,
  "next": null,
  "previous": null,
  "results": [
    {
      "title": "Piratas del Caribe: El cofre del hombre muerto",
      "average": 0,
      "collection": null,
      "movie_lang_id": 6136,
      "image": "/1c/3e/1c3e15050ceaf0e4edb15cdfb3c35867.jpg",
      "id": 3086
    },
    {
      "title": "Piratas del Caribe: En el fin del mundo",
      "average": 0,
      "collection": null,
      "movie_lang_id": 6138,
      "image": "/52/67/52678bde4104c39aed70938d1d82dc1d.jpg",
      "id": 3087
    },
    {
      "title": "Piratas del Caribe: La maldición de la Perla Negra",
      "average": 0,
      "collection": {
        "typeMovie": "seen",
        "id": 854
      },
```

```
    "movie_lang_id": 6140,
    "image": "/1d/8f/1d8f6066a9ad1cb91211bcbfca5915f2.jpg",
    "id": 3088
   },
   {
    "title": "Piratas del Caribe: En mareas misteriosas",
    "average": 0,
    "collection": null,
    "movie_lang_id": 12013,
    "image": "/5c/a0/5ca0ed89b7f45de6c912029b20151400.jpg",
    "id": 6042
   },
   {
    "title": "Piratas del Caribe: Los hombres muertos no cuentan cuentos",
    "average": 0,
    "collection": null,
    "movie_lang_id": 39934,
    "image": "/15/09/15092393d1f215f1769a50d5f97118fa.jpg",
    "id": 20813
   },
   {
    "title": "Lego Pirates of the Caribbean: The Video Game",
    "average": 0,
    "collection": null,
    "movie_lang_id": 74761,
    "image": "EXTERNAL#pelis/WN5C4474VZWTCUN",
    "id": 40053
   }
  ]
}
```


Obtenemos la información necesaria de una o varias películas para mostrar una previsualización en un listado. Nos devuelve su título en el idioma seleccionado, la carátula (*cover*) de la película, la puntuación de nuestro sistema, su id asociada a nuestra base de datos y su tipo de colección.

5.5 Script

Parte de la complejidad de este trabajo de fin de grado residía en la necesidad de actualizar la información sobre la puntuación de las películas. Esta información se ha obtenido a través de otras APIs o sitios web. Por esta razón, hemos desarrollado, durante gran parte del proyecto, un módulo de scripts y scrappers que nos permiten realizar esta labor.

En primer lugar explicaremos cómo hemos estructurado nuestros scripts en diferentes métodos y módulos de Python para que las tareas de mantenimiento del código sean menos complejas.

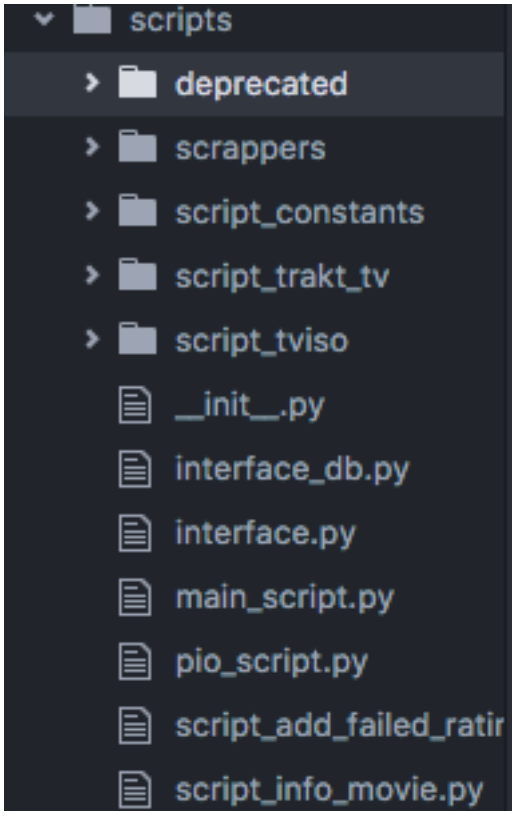

Figura 5.8. Directorio de scripts

Como script principal esta main_script que es el orquestador que hace referencia a todos los demás, y que va realizando las llamadas pertinentes a cada uno de los métodos necesarios. Por otra parte, tenemos interface_db e interface, el primero se encarga de gestionar contenidos y métodos de la base de datos mediante la clase DB, que nos permite conectarnos a la base de datos. El componente interface se encarga de guardar el estado, informar mediante logs y enviar correos de información.

Haciendo uso de paquetes de Python como http_client, podemos realizar, entre otras cosas, conectarnos al puerto del servidor que nos permite hacer las peticiones a la API.

```
  def __init__(self,user,password):

    self.connection = http.client.HTTPConnection("127.0.0.1",8080)

    self.headers = { "Authorization" : "Token " + "f490e66764821709149a760265da5e9b0c4a7cc1",

          "Content-type": "application/json"}
```

Para hacer las peticiones a la API usamos el módulo urllib.request y json.

```
def insert_data(self,api_url, js):
    self.connection.request('POST', api_url, js, self.headers)
    res = self.connection.getresponse()
    data = res.read().decode("utf8")
    return json.loads(data)
```

Como podemos observar, el request nos permite elegir el método de petición y cuándo decodificamos los datos obtenidos los serializamos a json para enviárselo a la API.

Para obtener la información troncal de la aplicación hacemos uso de APIs de terceros. Más concretamente de Tviso, para el idioma español, y Trakt.tv para el inglés. Por ello hemos creado diferentes directorios para encapsular los diferentes tipos de llamadas que realizan.

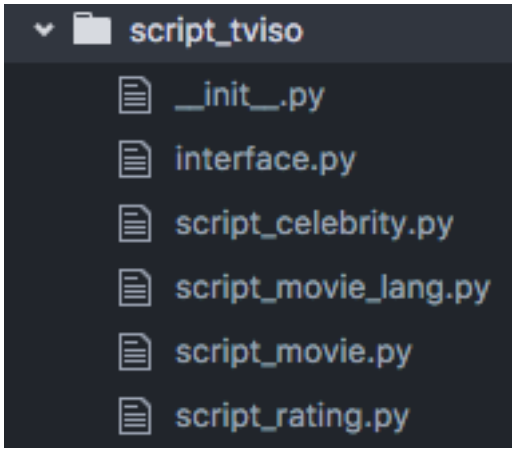

Figura 5.9. Directorio scripts de Tviso

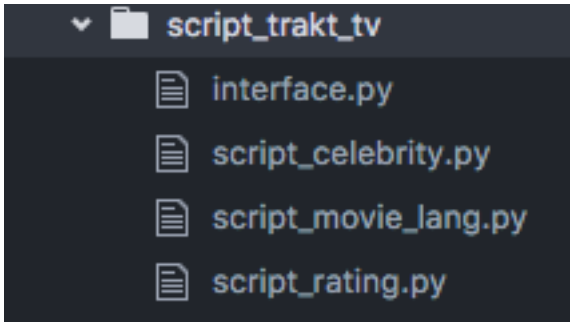

Figura 5.10. Directorio de scripts de Trakt.tv

Por otra parte, tenemos una fase de consulta de constantes que solo se realiza la primera vez. En este proceso se obtienen datos tales como diferentes idiomas, géneros, países …

Cierta información hemos tenido que obtenerla haciendo *scrappers* o arañas web que rastrean información de una página en concreto. Haciendo uso de la librería *BeautifulSoup,* que nos permite guardar el código html y mediante métodos de la librería como *find(),* buscamos la etiqueta html donde se encuentra la información que necesitamos.

```
def get_rating(soup):
    error_message = ""
    try:
        rating = soup.find(itemprop="ratingValue").get_text().strip()
        count = soup.find(itemprop="ratingCount").get_text().strip()
```

```
        rating = int(rating.replace(".", ""))
        count = int(count.replace(",", ""))
    except:
        error_message = "Error get rating IMDb\n"
        rating = 0
        count = 0

    return error_message, rating, count
```

Como podemos observar en este trozo de código, estamos buscando la puntuación en la web de IMDb. En ella esta información se encuentra en el elemento llamado “ratingValue”; lo parseamos y transformamos al tipo de dato que queremos y lo devolvemos en el *return* de la función.

Capítulo 6. Conclusiones y trabajo futuro

6.1 Conclusiones

En este proyecto se han descrito todas las fases necesarias para poder desarrollar una aplicación funcional multidispositivo nativa con diferentes tecnologías y arquitecturas. Esto nos ha permitido aprender tanto lenguajes de servidor como Django, lenguajes de creación de interfaces como HTML, CSS y JavaScript, y adicionalmente, estrategias de planificación y modelado de un proyecto.

Uno de las grandes utilidades de los sistemas de gestión de la información es precisamente facilitar el acceso a los datos. En nuestro caso la posibilidad de acceder y mantener el catálogo de películas de diversos usuario de forma centralizada, así como consultar y descubrir nuevas.

Haciendo uso de los frameworks Django y Django Rest, se han simplificado y agilizado muchas tareas que suelen ser tediosas para cualquier desarrollador. Entre estas podemos destacar el desarrollar una API en un servidor para acceder a la información de nuestra base de datos, así como el proceso de creación de las entidades de la base de datos, etc. Por otro lado, el sistema de plantillas de Django facilita enormemente la construcción de las páginas web. Finalmente, destacar también el proceso de construcción inicial de la estructura del proyecto, el cual se puede llevar a cabo con tan solo dos comandos ejecutados en un terminal.

Este trabajo de fin de grado se ha llevado a cabo en un equipo de tres personas. Cada miembro del grupo ha desarrollado una aplicación de cada plataforma, lo que nos ha permitido también aprender a trabajar en grupo en las partes comunes, como el desarrollo inicial de la base de datos y el modelado de la API, para posteriormente proceder a realizar un desarrollo en iteraciones o sprints para cada tipo de aplicación.

Como conclusión personal, durante este proyecto, he podido poner a prueba mis conocimientos en algunas materias que desconocía. En primer lugar, he podido descubrir el tipo de desarrollo que más me motiva de cara al futuro. En segundo lugar, en el proceso de desarrollo he aprendido las diferentes partes de la construcción de una aplicación web moderna teniendo en cuenta conceptos como arquitectura, diseño, modelado, mantenimiento, refactorización, etc. En resumen, ha sido una experiencia bastante gratificante en la que se ha desarrollado una aplicación completa y con actualización de contenidos automatizada funcional y usable.

He podido disfrutar de trabajar en equipo junto a mis compañeros en un problema real y solucionarlo usando la tecnología, además de poder ver cómo la gente hacia uso de él y mostraba especial interés en usarlo de forma continuada.

6.2 Trabajo Futuro

- **Sistema de recomendaciones:** Parte de la gran importancia de este tipo de aplicaciones se basa en dar a conocer nuevas obras culturales del cine en base a los gustos de cada persona. Por ello una funcionalidad podría ser la implementación de un sistema de recomendaciones basado, por ejemplo, en técnicas de Aprendizaje Automático (Machine Learning).
- **Inicio de sesión y registro con redes sociales:** Gran parte de la interacción hoy en día con el registro e inicio de sesión en las aplicaciones actuales pasa por el uso de las redes sociales más utilizadas hoy en día como son Twitter, Facebook o Google.
- **Compartir en redes sociales:** Como hemos comentado antes, la importancia de compartir tus acciones en redes sociales.
- **Validación por correo:** Un paso muy importante del registro de un usuario es la verificación por correo.
- **Interfaz de usuario con Angular JS:** Realizar el desarrollo de la capa cliente con Angular JS nos permitirá un desarrollo más ágil e interactivo de la interfaz de usuario. Esto facilitará el renderizado automático de la vista, haciendo más cómoda la visualización de cara al usuario.
- **Seguimiento a celebridades:** Gran parte del interés de una obra cinematográfica son sus actores (o directores) más relevantes. Muchas personas emplean como criterio a la hora de seleccionar una película, los actores o directores que les gustan. Por ello una funcionalidad sería poder seguir a esa celebridad y ver en qué películas ha participado.
-

Bibliografía

[1] Bahit, E. (2012). *Curso: Python para principiantes.* Buenos Aires, Argentina: Safe creative.

[2] Beati, H. (2015). *HTML5 y CSS3 para diseñadores.* Argentina: Alfaomega Grupo Editor Argentino.

[3] Christie, T. (2011-2016). *Django REST framework*. Obtenido de Django REST framework: http://www.django-rest-framework.org

[4] Django Software Foundation. (2015-22016). *The Web framwetk for perfectionists with deadlines* | *Django*. Obtenido de Django: https://www.djangoproject.com

[5] Github. (2016). Obtenido de https://github.com

[6] M., S. G. (2015). *La guia definitiva de Django: Desarrolla aplicaciones Web de forma rápida y sencilla.* Celayita, México: Django Software Corporation.

[7] Richardson, C. a. (2016). *Beautiful Soup: We called him Tortoise because he taught us.* Obtenido de Beautiful Soup: https://www.crummy.com/software/BeautifulSoup/

[8] Tviso. (2015). *Tviso API*. Obtenido de Tviso: https://developers.tviso.com

[9] Twitter, bootstrap. (2010). *Bootstrap: The world´s most popular mobile-firts and responsive front-end framework*. Obtenido de Bootstrap: http://getbootstrap.com

[10] W3Schools Online Web Tutorials. (2016). Obtenido de W3School: http://www.w3schools.com