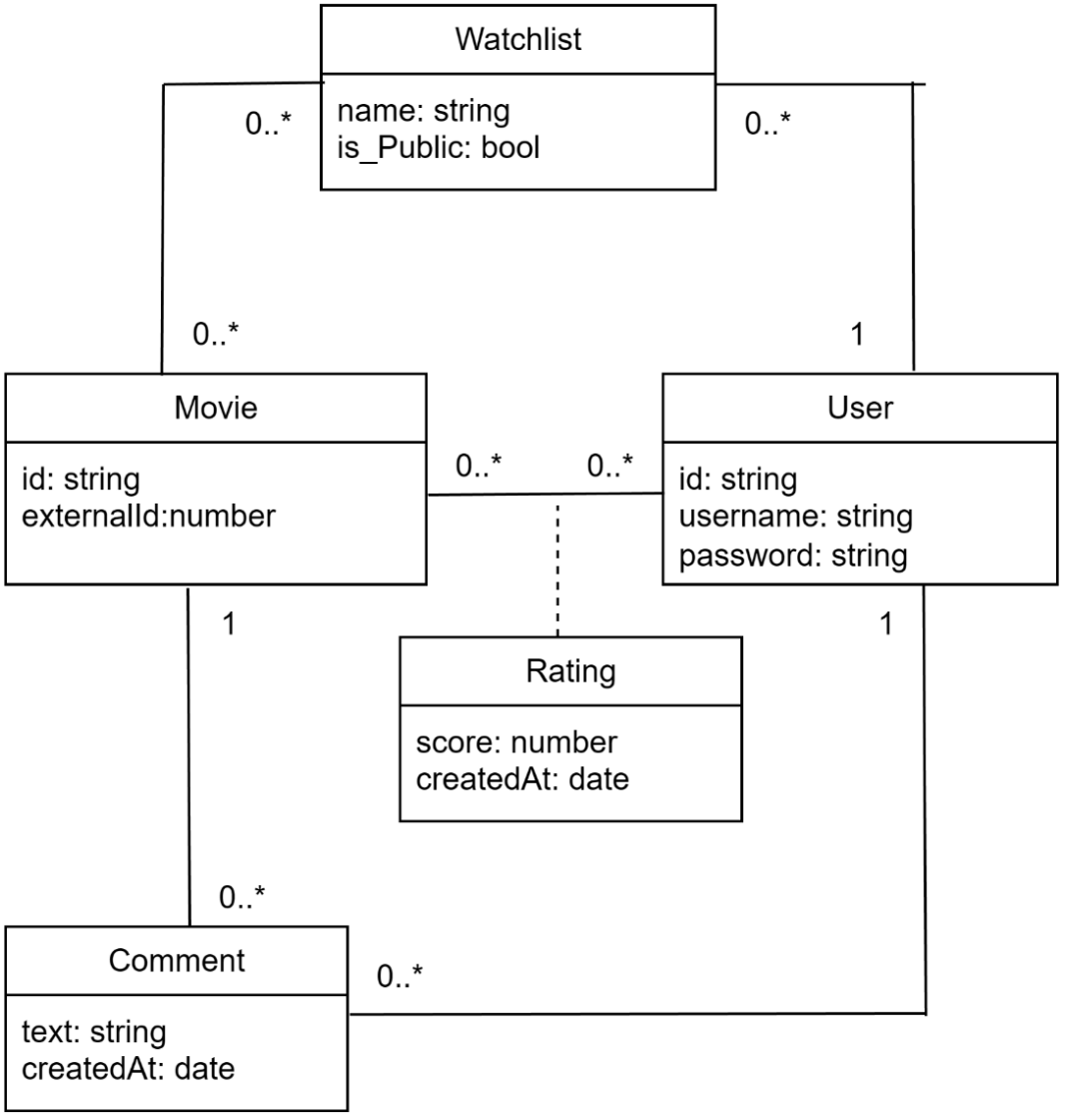
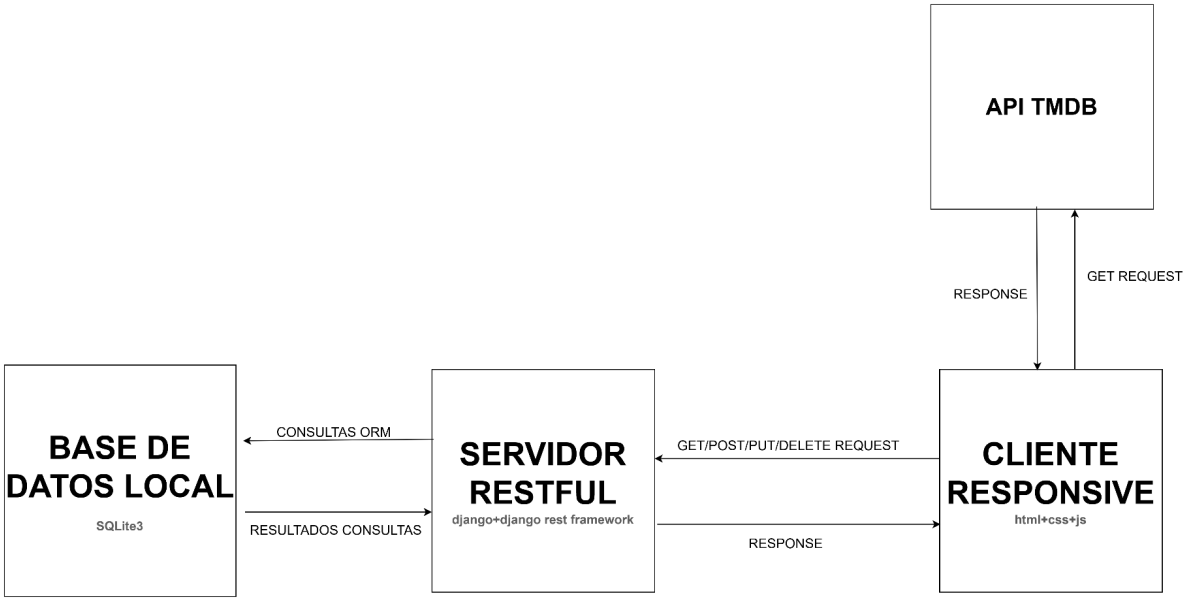
Documentación trabajo final de PMUD

Jesika Jiménez

1. UML



2. Diagrama de la estructura modular



El proyecto se organiza siguiendo una **arquitectura cliente–servidor**, con una clara separación de responsabilidades entre la interfaz de usuario, la lógica de negocio y la persistencia de datos. Esta estructura modular facilita el mantenimiento, la escalabilidad y la reutilización del código.

1. Cliente Responsive (Frontend)

El **cliente responsive** es el módulo encargado de la interacción con el usuario. Está desarrollado utilizando **HTML, CSS y JavaScript**, permitiendo una adaptación correcta a diferentes tamaños de pantalla y dispositivos.

Sus principales responsabilidades son:

- Renderizar la interfaz gráfica.
- Gestionar la navegación y eventos del usuario.
- Enviar solicitudes HTTP al servidor RESTful.
- Procesar las respuestas recibidas en formato JSON.

La comunicación con el servidor se realiza mediante **peticiones REST** utilizando los métodos **GET, POST, PUT y DELETE**, siguiendo el protocolo HTTP.

Este módulo **no accede directamente a la base de datos**, lo que garantiza un bajo acoplamiento y una mayor seguridad.

2. Servidor RESTful (Backend)

El servidor constituye el núcleo de la aplicación y está desarrollado con **Django** y **Django REST Framework (DRF)**. Su función principal es exponer una **API REST** que encapsula la lógica de negocio del sistema.

Este módulo se estructura internamente en varias capas:

- **Vistas / ViewSets**: gestionan las peticiones HTTP entrantes y devuelven las respuestas adecuadas.
- **Serializers**: transforman los datos entre objetos Python y representaciones JSON.
- **Modelos**: definen la estructura de los datos y las relaciones entre las entidades.
- **URLs**: enrutan las peticiones hacia los controladores correspondientes.
- **Autenticación y permisos**: controlan el acceso a los recursos del sistema.

El servidor actúa como intermediario entre el cliente y la base de datos, asegurando que todas las operaciones cumplan las reglas de negocio definidas.

3. Base de Datos Local

La persistencia de datos se gestiona mediante una **base de datos local SQLite**, integrada con Django.

El acceso a la base de datos se realiza exclusivamente a través del **ORM de Django**, que permite ejecutar operaciones **CRUD (Create, Read, Update, Delete)** de forma abstracta, sin necesidad de escribir consultas SQL directamente.

Este enfoque aporta:

- Independencia del motor de base de datos.

- Mayor seguridad frente a inyecciones SQL.

- Facilidad para modificar el esquema mediante migraciones.

4. Comunicación entre módulos

La comunicación entre los módulos se realiza de la siguiente manera:

- **Cliente ↔ Servidor RESTful:** Mediante peticiones HTTP REST (GET, POST, PUT, DELETE) con intercambio de datos en formato JSON.

- **Servidor RESTful ↔ Base de Datos:** A través del **Django ORM**, encargado de gestionar las operaciones de acceso y persistencia de datos.

Esta separación garantiza una arquitectura desacoplada, en la que cada módulo cumple una función bien definida.

TABLA COMPLETA DE RUTAS API

Autenticación y Usuarios

Ruta	Método	Descripción	Parámetros	Respuestas
`/api/register/`	POST	Registrar un nuevo usuario	`{ username, password }` (body)	201: `{token, user_id, username}` 400: Errores de validación
`/api/login/`	POST	Iniciar sesión	`{ username, password }` (body)	200: `{token, user_id, username}` 400: Credenciales inválidas
`/api/logout/`	POST	Invalidar token (simulado)	Token en header	200: Token invalidado
`/api/users/`	GET	Listar todos los usuarios	Token en header	200: Lista de usuarios 401: No autenticado
`/api/users/`	POST	Crear nuevo usuario (admin)	`username`, `password` (body)	201: Usuario creado 400: Errores de validación 401: No autenticado
`/api/users/{id}/`	GET	Obtener usuario específico	`id` (path)	200: Detalles del usuario 404: Usuario no encontrado
`/api/users/{id}/`	PUT	Actualizar usuario completo	`id` (path), `username`, `password` (body)	200: Usuario actualizado 400: Errores 404: No encontrado
`/api/users/{id`	PATCH	Actualizar parcialmente	`id` (path),	200: Usuario actualizado

`}/`		usuario	campos a actualizar (body)	400: Errores 404: No encontrado
`/api/users/{id}/`	DELETE	Eliminar usuario	`id` (path)	204: Eliminado 404: No encontrado
`/api/users/me/`	GET	Obtener información del usuario actual	`Token en header`	200: Usuario actual 401: No autenticado

Películas (Movies)

Ruta	Método	Descripción	Parámetros	Respuestas
`/api/movies/`	GET	Listar todas las películas	Token en header `externalId` (query, opcional)	200: Lista de películas 401: No autenticado
`/api/movies/`	POST	Crear nueva película	Token en header, `externalId` (body)	201: Película creada 400: Errores de validación 401: No autenticado
`/api/movies/{id}/`	GET	Obtener película específica	`id` (path, UUID)	200: Detalles de película 404: No encontrada
`/api/movies/{id}/`	PUT	Actualizar película completa	`id` (path), `externalId` (body)	200: Película actualizada 400: Errores 404: No encontrada
`/api/movies/{id}/`	PATCH	Actualizar parcialmente película	`id` (path), campos a actualizar (body)	200: Película actualizada 400: Errores 404: No encontrada

`/api/movies/{id}/`	DELETE	Eliminar película	`id` (path)	204: Eliminada 404: No encontrada

Watchlists (Listas de seguimiento)

Ruta	Método	Descripción	Parámetros	Respuestas
`/api/watchlists/`	GET	Listar watchlists disponibles	Token en header	200: Lista de watchlists (propias + públicas) 401: No autenticado
`/api/watchlists/`	POST	Crear nueva watchlist	Token en header, `name`, `isPublic` (body)	201: Watchlist creada 400: Errores de validación 401: No autenticado
`/api/watchlists/{id}/`	GET	Obtener watchlist específica	`id` (path, UUID)	200: Detalles de watchlist 403: No permiso (si es privada y no es propietario) 404: No encontrada
`/api/watchlists/{id}/`	PUT	Actualizar watchlist completa	`id` (path), `name`, `isPublic` (body)	200: Watchlist actualizada 403: No es propietario 404: No encontrada
`/api/watchlists/{id}/`	PATCH	Actualizar parcialmente watchlist	`id` (path), campos a actualizar (body)	200: Watchlist actualizada 403: No es propietario 404: No encontrada
`/api/watchlists/{id}/`	DELETE	Eliminar watchlist	`id` (path)	204: Eliminada 403: No es propietario 404: No encontrada
`/api/watchlists/{id}/movies/`	GET	Obtener películas de una watchlist	`id` (path)	200: Lista de películas 403: No permiso (si

				es privada y no es propietario) 404: No encontrada
`/api/watchlists/{id}/add_movie/`	POST	Añadir película a watchlist	`id` (path), `movieId` (body, UUID)	201: Película añadida 400: Ya existe 403: No es propietario 404: Watchlist o película no encontrada

Relaciones Watchlist-Movie

Ruta	Método	Descripción	Parámetros	Respuestas
`/api/watchlist-movies/`	GET	Listar relaciones watchlist-película	Token en header `watchlist` (query, opcional) `movie` (query, opcional	200: Lista de relaciones 401: No autenticado
`/api/watchlist-movies/`	POST	Crear relación (añadir película a watchlist)	Token en header, `watchlistId`, `movieId` (body, ambos UUID)	201: Relación creada 400: Ya existe o errores 403: No es propietario de watchlist 404: Watchlist o película no encontrada
`/api/watchlist-movies/{id}/`	GET	Obtener relación específica	`id` (path, UUID)	200: Detalles de relación 404: No encontrada
`/api/watchlist-movies/{id}/`	PUT	Actualizar relación completa	`id` (path), `watchlistId`, `movieId` (body)	200: Relación actualizada 400: Errores 404: No encontrada
`/api/watchlist-movies/{id}`	PATCH	Actualizar parcialmente	`id` (path), campos a	200: Relación actualizada

`/`		relación	actualizar (body)	400: Errores 404: No encontrada
`/api/watchlist-movies/{id}/`	DELETE	Eliminar relación (quitar película de watchlist)	`id` (path)	204: Eliminada 403: No es propietario de watchlist 404: No encontrada
`/api/watchlist-movies/by_watchlist/`	GET	Obtener películas de watchlist específica	`watchlist` (query, UUID, requerido)	204: Eliminada 403: No es propietario de watchlist 404: Watchlist no encontrada
`/api/watchlist-movies/by_movie/`	GET	Obtener watchlists que contienen una película	`movie` (query, UUID, requerido)	200: Lista de relaciones 404: Película no encontrada

Ratings (Calificaciones)

Ruta	Método	Descripción	Parámetros	Respuestas
`/api/ratings/`	GET	Listar calificaciones	Token en header `movie` (query, opcional) `userId` (query, opcional	200: Lista de calificaciones (propias por defecto) 401: No autenticado
`/api/ratings/`	POST	Crear/actualizar calificación	Token en header, `movie_uuid`, `score` (1-5) (body)	201: Calificación creada 200: Calificación actualizada 400: Errores de validación 401: No autenticado
`/api/ratings/{id}/`	GET	Obtener calificación específica	`id` (path, UUID)	200: Detalles de calificación 404: No encontrada

`/api/ratings/{id}/`	PUT	Actualizar calificación completa	`id` (path), `score` (1-5) (body)	200: Calificación actualizada 400: Errores 404: No encontrada
`/api/ratings/{id}/`	PATCH	Actualizar parcialmente calificación	`id` (path), `score` (1-5) (body)	200: Calificación actualizada 400: Errores 404: No encontrada
`/api/ratings/{id}/`	DELETE	Eliminar calificación	`id` (path)	204: Eliminada 404: No encontrada

Comentarios (Comments)

Ruta	Método	Descripción	Parámetros	Respuestas
`/api/comments/`	GET	Listar comentarios	Token en header `movie` (query, opcional) `userId` (query, opcional)	200: Lista de comentarios 401: No autenticado
`/api/comments/`	POST	Crear nuevo comentario	Token en header, `movie_uuid`, `text` (body)	201: Comentario creado 400: Errores de validación 401: No autenticado
`/api/comments/{id}/`	GET	Obtener comentario específico	`id` (path, UUID)	200: Detalles de comentario 404: No encontrado
`/api/comments/{id}/`	PUT	Actualizar comentario completo	`id` (path), `text` (body)	200: Comentario actualizado 400: Errores 403: No es propietario 404: No encontrado
`/api/comments/{id}/`	PATCH	Actualizar parcialmente comentario	`id` (path), campos a actualizar	200: Comentario actualizado 403: No es

			(body)	propietario 404: No encontrado
`/api/comments/{id}/`	DELETE	Eliminar comentario	`id` (path)	204: Eliminado 403: No es propietario 404: No encontrado

Notas Importantes:

Autenticación:

- Todas las rutas (excepto `/register/` y `/login/`) requieren Token de autenticación en el header:
  text
  Authorization: Token <tu_token_aqui>

Parámetros:

- Path parameters: Van en la URL (`{id}`)
- Query parameters: Van después de `?` en la URL (`?movie=uuid&userId=1`)
- Body parameters: Se envían en el cuerpo de la petición (JSON)

UUIDs:

- Todos los IDs (excepto usuarios) son UUIDs, no números enteros

Permisos:

- Usuarios solo pueden modificar/eliminar sus propios recursos (watchlists, comentarios, etc.)
- Watchlists privadas solo son visibles para su propietario

Códigos de estado comunes:

- 200: OK
- 201: Creado
- 204: Eliminado (sin contenido)
- 400: Error en la solicitud
- 401: No autenticado
- 403: Prohibido (no tiene permisos)
- 404: No encontrado
- 500: Error interno del servidor