FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Ejercicio: Diseño de API REST y Arquitectura para una App de Películas

Ingenieria de Software I
Lucas Araujo - Padron : 109867

Índice

1. Introducción

En este ejercicio, se diseña una API REST para una aplicación móvil que permite a los usuarios interactuar con una base de datos de películas y realizar diversas acciones sociales, como seguir a otros usuarios y calificar películas. El objetivo es que apliques buenas prácticas de desarrollo de software, tanto en el diseño de la API como en la definición del modelo de dominio y la arquitectura preliminar de la aplicación.

2. Arquitectura basica preliminar

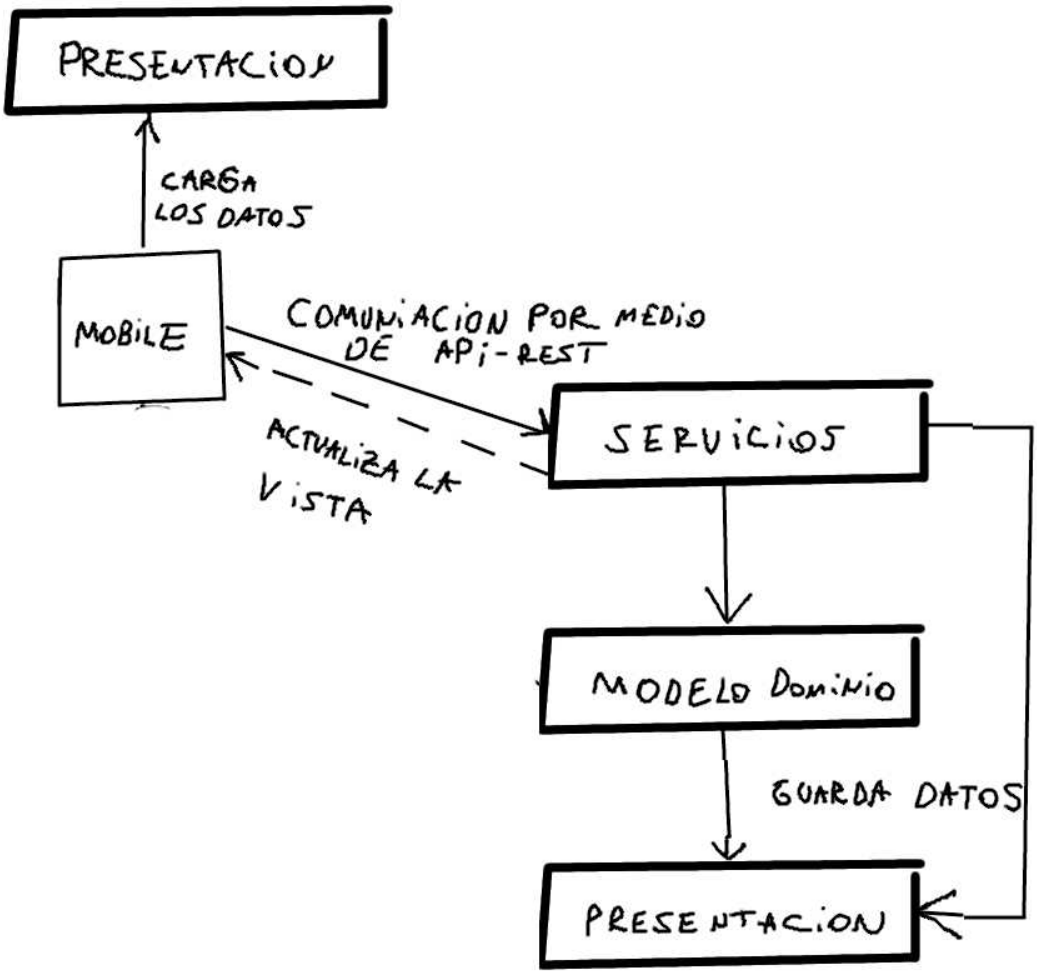

Figura 1: Diagrama basico para explicar la arquitectura

Se ha decidido utilizar la arquitectura "layers"para el sistema. La primera capa es la de Presentación, donde el usuario, a través de la aplicación móvil, carga los archivos correspondientes y puede interactuar mediante una API REST con los servicios. Estos servicios proponen una interfaz con diferentes endpoints, donde se realizará la verificación de la autorización del usuario y demas precondiciones para pasar a la capa del modelo de dominio.

La capa de Modelo de Dominio se encargará de gestionar el caso de uso solicitado, siempre y cuando se hayan cumplido las precondiciones en la capa de servicio. Esta capa se centra en la logica central del sistema y tiene acceso a los datos de la capa de Persistencia.

Ademas, existen servicios que no requieren pasar por el modelo de dominio, como un GET de las peliculas. Por esta razon, la capa de Servicios tiene una asociacion directa con la capa de

Persistencia.

Por ultimo el servicio devuelve una respuesta y en base a esa respuesta se actualiza la vista de la UI del telefono si es necesario.

3. Modelo de dominio preliminar

El modelo del dominio preliminar se captaron las entidades principales que son. **Pelicula, Categoria, Persona, Actores. Con estas entidades atacaran los casos de usos requeridos por el sistema.** Se mostrara un diagrama de clases para explicar mejor el modelo de dominio.

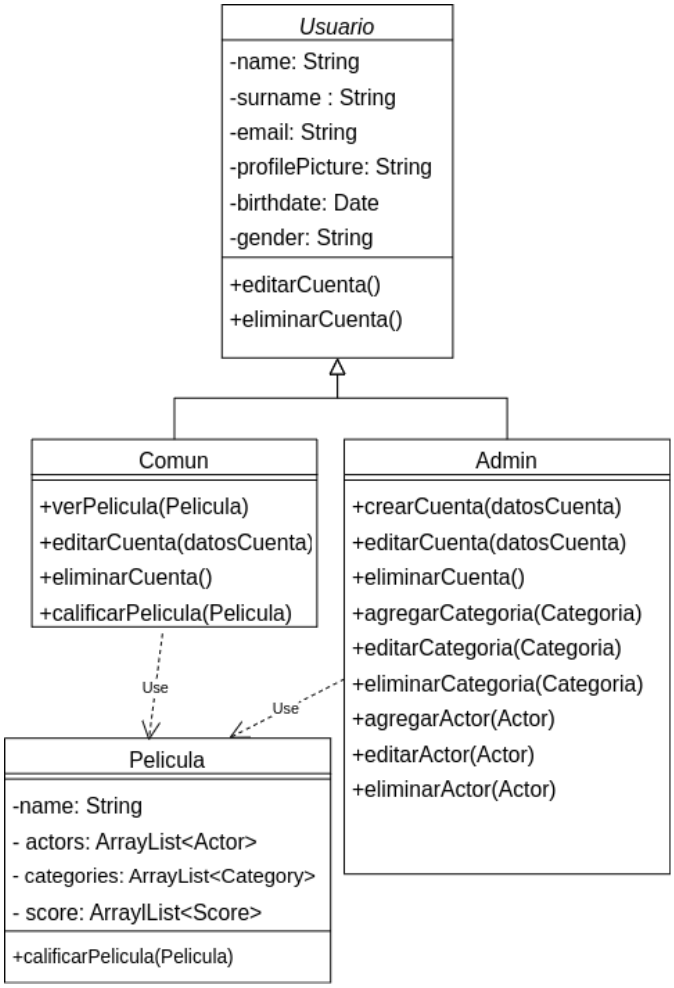

Figura 2: Diagrama de clases para explicar mejor el modelo de dominio

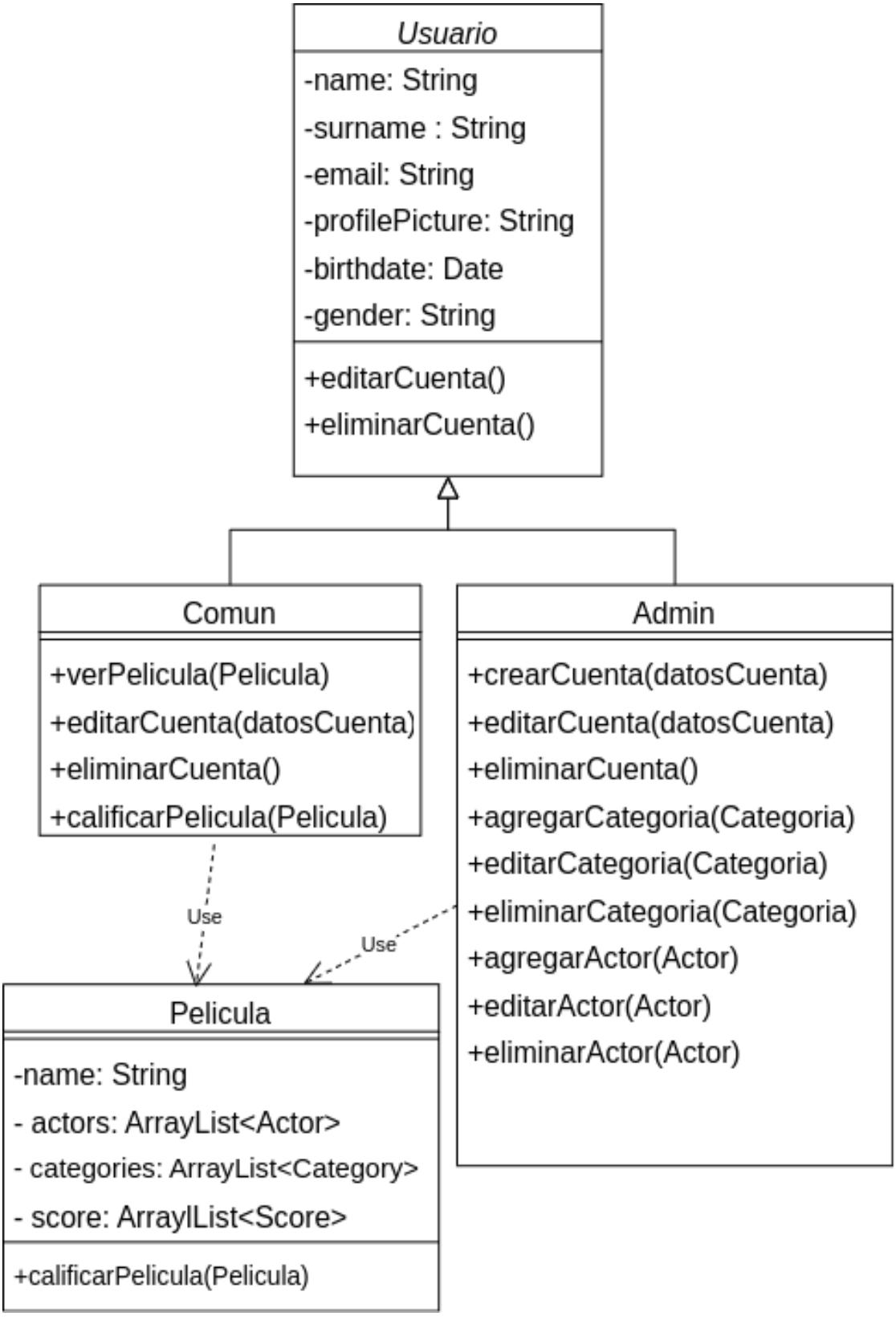

Figura 3: Esto es el caption

4. Endpoints de la aplicacion

4.1. Rol sin estar logueado

4.1.1. Buscar Peliculas

```
Request:
    GET /api/v1/films?criterios="nombre_criterio"

Response: 200 OK
{
        "peliculas": [{
        "pelicula": {
            "name": "nombre de la pelicula",
            "description": "descripcion de la pelicula",
            "actors": "actores" : [{...}],
            ... etc
        }
    }
    ... el resto de peliculas
    ]
}

Código HTTP: 400 Bad Request si el nombre del criterio no existe
```

Comentarios: En la peticion /films?criterio, criterio puede ser filtros por título, actores y categorías. El criterio lo modifica el usuario a traves de la capa de presentacion y el servidor procesa y envia los datos con una lista de peliculas que cumplan con el criterio correspondiente

4.2. Rol estando logueado

4.2.1. Calificar peliculas con puntuacion del 1 al 10

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld
    POST /api/v1/films/{id}/ratings
    Body{
    "score": 8
    }

Response: 200 OK
{
        "message": "Actor editado correctamente",
}

Código HTTP: 400 Bad Request si el id de la pelicula no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.2.2. Buscar otros usuarios

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    GET /api/v1/users?page=1&size=50

Response: 200 OK
{
        users:[{...Lista con todos los usuarios del 1 al 50}]
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Se utilizo paginacion para que no me devuelva los miles de usuarios en una sola peticion asi no hacemos que el servidor colapse. Tambien deberiamos dejar que el usuario busque por id al usuario.

4.2.3. Buscar otros usuarios por id

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    GET /api/v1/users/{id}

Response: 200 OK
{
        user: usuario: {informacion basica del usuario...}
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: La respuesta del servidor es la informacion basica del usuario es decir que el servidor le manda el id, nombre, apellido, la foto de perfil, alguna cosita mas del usuario, pero, **NO** le envia la informacion privada del usuario como la contraseña, email, numero de telefono, direccion de la casa, etc

4.2.4. Seguir a los Usuarios

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    POST /api/v1/users/{id}/follow

Response: 200 OK
{
        user: usuario: {informacion basica del usuario...}
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Se le envia en la peticion POST el id del usuario que quiero seguir. El servidor le responde con la infromacion basica del usuario, esta respuesta no es para agregar a la base de datos, sino para confirmar que la accion se ha realizado correctamente. La logica que agrega a la base de datos la hace el serivdor en el endpoint que definimos /api/v1/users/id/follow

4.2.5. Ver las peliculas que han calificado los usuarios a los que sigo

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    GET /api/v1/users/follows?page=1&size=50

Response: 200 OK
{
        users: [user: {informacion del score de los usuarios de las peliculas.}]
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Aca se podria hacer de muchas formas dependiendo el requerimiento. Una forma seria que se le mande la informacion del score de todos los usuarios a los que sigo de alguna pelicula especifica o ver todos los score de peliculas de un usuario especifico.

4.2.6. Aceptar o rechazar solicitudes de seguimiento.

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    POST /api/v1/users/{id}/follows-request/{requestId/accept}
    para el caso de aceptar
```

```
    POST /api/v1/users/{id}/follows-request/{requestId}/reject
    para el caso de rechazar

Response: 200 OK
{
        "message": "Solicitud de seguimiento aceptada/rechazada.",
    "user": {
        "id": 123,
        "name": "Nombre de la persona que se le empezo a seguir"
    }
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: el id es el id del usuario que esta aceptando o rechazando la solicitud (esta informacion se encuentra presente en el token jwt) y el requestId es el el id del usuario que le mando solicitud

4.2.7. Ver su lista de seguidores y a quién está siguiendo.

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    GET /api/v1/users/{id}/followers?page=1&size=50 para el caso de ver su lista de seguidores

    GET /api/v1/users/{id}/following?page=1&size=50 para el caso de ver a quien esta siguiendo

Response: 200 OK
{
    "users": [{
        "id": 123,
        "name": "Nombre del seguidor/seguido",
        etc...
    },
    ...el resto de usuarios
    ]
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: el id es la el id del usuario autenticado, esta informacion esta en el token jwt.

4.2.8. Ver Perfil

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    GET /api/v1/users/{id}/profile

Response: 200 OK
{
    "user": {
        "id": 123,
        "email": "email",
        "name": "nombre",
        "surname": "apellido",
        "profile_picture": "foto de perfil",
        "date_of_birth": "fecha de nacimiento",
        "gender": "genero"
    },
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Si el usuario id tiene la autorizacion se le muestra los detalles del perfil de el mismo.

4.2.9. Editar Perfil

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    PUT /api/v1/users/{id}/profile
    Body:{
        "email": "email",
        "name": "nombre",
        "surname": "apellido",
        "profile_picture": "foto de perfil",
        "date_of_birth": "fecha de nacimiento",
        "gender": "genero"
    }
Response: 200 OK
{
    message: "El usuario se ha modificado correctamente"
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Todos estos campos pueden ser modificados o quedar como estaban antes. Aca tenemos muchas formas de editar una forma seria editar todo de una como lo planteamos en este request (Un ejemplo seria los tipicos formularios que editas los campos y pones "guardar todo"). Otra forma podria ser editar por cada dato especifico. por ejemplo podriamos tener varias rutas de edicion tales como ...id/profile/name para editar el nombre o ...id/profile/email para editar el email, etc.

4.2.10. Eliminar Perfil

```
Request:
    Headers: Authorization: token jwt feJhIUzI1NiIsInR5cVCJ9.eyJ1aRld}
    DELETE /api/v1/users/{id}

Response: 204 NO CONTENT
{
    message: "El usuario se ha eliminado correctamente"
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.3. Rol Admin

4.3.1. Descripción: Iniciar sesion como administrador.

```
Request:
    Headers: Authorization: Basic{base64(email:password)}
    POST /api/v1/login/admins/token

Response: 200 OK
{
 ‘‘access_token’’: ‘‘eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ’’
 ‘‘refresh_token’’: ‘‘feJsbUze2DmnsInRpXVCR9.eyJ1avzL’’
        "message": "Usuario admin logueado correctamente"
}

Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: El access token esta dividida en tres partes con informacion util para el servidor. El primer tercio del token contiene la data del tipo de token y el tipo de algoritmo que se utilizo

para hashearlo, la segunda parte tendria infromacion del usuario (admin) y el ultimo segmento tendria la informacion de como es la firma por parte del servidor lo que le hace muy seguro ya que solo el servidor sabe su propia firma.

4.3.2. Descripción: Crear otros usuarios de rol Admin

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    POST /api/v1/admins/register
    Body:{
        "email": "email@fi.uba.ar",
        "password": "password del usuario",
        "name": "nombre",
        "surname": "apellido",
        "profile_picture": "foto de perfil",
        "birthdate": "1/1/2001",
        "gender:"masculino"
    }
Response: 201 CREATED
{
        "message": "Usuario admin creado correctamente",
}

Código HTTP: 400 Bad Request si el email ya existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Si la peticion es correcta devuelve un mensaje que el usuario admin se creo correctamente y guarda en la base de datos el usuario con todos sus datos. Solo debe poder crear un usuario admin aquel que tenga el token proveniente de un usuario admin, la infromacion de si el usuario es admin o no se encuentra en el mismo token jwt.

4.3.3. Descripción: Editar otros usuarios de rol Admin

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    PUT /api/v1/admins/{id}
    Body:{
        "email": "email@fi.uba.ar",
        "name": "nombre",
        "surname": "apellido",
        "profile_picture": "foto de perfil",
        "birthdate": "1/1/2001",
        "gender:"masculino"
    }

Response: 200 OK
{
 ‘‘access_token’’: ‘‘eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ’’,
 ‘‘refresh_token’’: ‘‘feJsbUze2DmnsInRpXVCR9.eyJ1avzL’’,
        "message": "Usuario se modifico correctamente",
}

Código HTTP: 400 Bad Request si el email o id no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Al igual que para crear Necesito el token del Admin

4.3.4. Descripción: Eliminar otros usuarios de rol Admin

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    DELETE /api/v1/admins/{id}
```

```
Response: 204 OK
{
        "message": "Se elimino el usuario correctamente"
}

Código HTTP: 400 Bad Request si el email o id no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: Al igual que para crear Necesito el token del Admin

4.3.5. Crear categorías de películas

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    POST /api/v1/films-categories
    Body{
    "category": "Nombre de la nueva categoria"
    }

Response: 201 CREATED
{
        "message": "Categoria creada correctamente",
}

Código HTTP: 400 Bad Request si el nombre de la categoria ya existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.3.6. Editar categorías de películas

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    PUT /api/v1/films-categories
    Body{
    category= "Categoria a editar"
    }

Response: 200 OK
{
        "message": "Categoria editada correctamente",
}

Código HTTP: 400 Bad Request si el nombre o id de la pelicula no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.3.7. ELiminar categorías de películas

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    DELETE /api/v1/categories-films
    Body{
    "category": "Nombre de la categoria a eliminar"
    }

Response: 204 NO CONTENT
{
        "message": "Categoria eliminada correctamente",
}

Código HTTP: 400 Bad Request si el nombre o id de la pelicula no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.3.8. Crear actores

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    POST /api/v1/actors
    Body{
    "name": "nombre de la actor",
    "surname": "apellido del actor",
    "picture": "foto del actor",
    ...etc
    }

Response: 201 CREATED
{
        "message": "Actor creado correctamente",
}

Código HTTP: 400 Bad Request si el nombre de la actor ya existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

Comentarios: El id no hace falta pasarlo porque la asignacion del id se encarga el servidor.

4.3.9. Editar actores

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    PUT /api/v1/actors/{id}
    Body{
    "name": "nombre de la actor",
    "surname": "apellido del actor",
    "picture": "foto de la actor",
    ...etc
    }

Response: 200 OK
{
        "message": "Actor editado correctamente",
}

Código HTTP: 400 Bad Request si el nombre o id de la actor no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.3.10. ELiminar categorías de películas

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    DELETE /api/v1/actors/{id}
    Body{
    "name": "nombre del actor"
    "surname": "apellido del actor",
    }

Response: 204 NO CONTENT
{
        "message": "Actor eliminado correctamente",
}

Código HTTP: 400 Bad Request si el nombre y apellido o id de la actor no existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```

4.3.11. Modificar peliculas, asignando actores y categorias (sin modificar las calificaciones de los usuarios

```
Request:
    Headers: Authorization: token jwt eyJhIUzI1NiIsInR5cVCJ9.eyJ1aQzQ}
    PATCH /api/v1/films/categories-actors
    Body:{
    categories:[{category:{...}},...]
    actors:[{actor:{...}},...]
    }

Response: 200 OK
{
        "message": "Actores y Categorias modificados correctamente",
}

Código HTTP: 400 Bad Request si el nombre de la actor ya existe.
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.
```