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

Alumna: Lara Daniela Converso
Padrón: 107632

1. Definir una API REST:

Login:
Descripción: Permite que los usuarios inicien sesión.
Request:
Headers: Authorization: Basic {base64(email:password)}
Endpoint: POST /api/v1/users/token
Response:
Código HTTP: 200 OK
{
"access_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
"refresh token": "m9obiBEb2UiLCJpYXQiOjE1MTYyMzkwMj..."
}
Código HTTP: 401 Unauthorized si las credenciales son incorrectas.

Crear usuario admin:
Descripción: Permite que un administrador cree otro usuario de rol ADMIN.
Request:
Headers: Authorization: Bearer {access_token}
Body:
{
"email": "usuario@example.com",
"nombre": "User1",
"apellido": "One",
"fecha_nacimiento": "1999-09-09",
"género": "Femenino",
"avatar": "url_profilepic",
"rol": "USER"
}
Endpoint: POST /api/v1/admin/users/register
Response:
Código HTTP: 200 OK//201 CREATED
{
"message": "Usuario creado exitosamente"
}
Código HTTP: 400 Bad Request si hay error en la validación de los datos.

Editar usuario ADMIN:

Descripción: Permite que un usuario edite a otro usuario de rol ADMIN**.**
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
  "nombre": "NuevoNombre",
  "apellido": "NuevoApellido",
  "género": "Femenino",
  "avatar": "url_nueva_foto"
}
```

Endpoint: PUT /api/v1/admin/users/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Usuario actualizado exitosamente"
}
```

Código HTTP: 404 Not Found si el usuario no existe.
Código HTTP: 400 Bad Request si hay error en la validación.

Eliminar usuario ADMIN:
Descripción: Permite que un administrador elimine un usuario.
Request:

Headers: Authorization: Bearer {access_token}
Endpoint: DELETE /api/v1/admin/users/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Usuario eliminado exitosamente"
}
```

Código HTTP: 404 Not Found si el usuario no existe.

Crear categorías de películas:
Descripción: Permite que un administrador cree una nueva categoría de películas.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
"nombre": "Acción"
}
```

Endpoint: POST /api/v1/admin/categories

Response:

Código HTTP: 200 OK // 201 CREATED

```
{
  "message": "Categoría creada exitosamente"
}
```

Código HTTP: 400 Bad Request si hay error en la validación

Editar categorías de películas:
Descripción: Permite que un administrador edite una categoría de películas.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
"nombre": "Drama"
}
```

Endpoint: PUT /api/v1/admin/categories/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Categoría actualizada exitosamente"
}
```

Código HTTP: 404 Not Found si la categoría no existe.

Eliminar categorías de películas:
Descripción: Permite que un administrador elimine una categoría de películas.
Request:

Headers: Authorization: Bearer {access_token}
Endpoint: DELETE /api/v1/admin/categories/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Categoría eliminada exitosamente"
}
```

Código HTTP: 404 Not Found si la categoría no existe.

Crear actores:
Descripción: Permite que un administrador cree un nuevo actor.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
  "nombre": "Hugh Jacman"
}
```

Endpoint: POST /api/v1/admin/actors

Response:

Código HTTP: 200 OK // 201 CREATED

```
{
  "message": "Actor creado exitosamente"
}
```

Código HTTP: 400 Bad Request si hay error en la validación.

Editar actores:

Descripción: Permite que un administrador edite un actor.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
  "nombre": "Nombre editado"
}
```

Endpoint: PUT /api/v1/admin/actors/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Actor actualizado exitosamente"
}
```

Código HTTP: 404 Not Found si el actor no existe.

Eliminar actores:
Descripción: Permite que un administrador elimine un actor.
Request:

Headers: Authorization: Bearer {access_token}
Endpoint: DELETE /api/v1/admin/actors/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Actor eliminado exitosamente"
}
```

Código HTTP: 404 Not Found si el actor no existe.

Crear películas:
Descripción: Permite que un administrador cree una nueva película.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
  "título": "Película Ejemplo",
  "año": 2024,
  "duración": 120,
  "categoría": "Acción",
  "actores": ["Actor 1", "Actor 2"]
}
```

Endpoint: POST /api/v1/admin/movies

Response:

Código HTTP: 200 OK // 201 CREATED

```
{
  "message": "Película creada exitosamente"
}
```

Código HTTP: 400 Bad Request si hay error en la validación.

Editar películas:
Descripción: Permite que un administrador edite una película existente.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
  "título": "Nueva Película",
  "año": 2025,
  "duración": 150,
  "categoría": "Drama",
  "actores": ["Actor 3", "Actor 4"]
}
```

Endpoint: PUT /api/v1/admin/movies/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Película actualizada exitosamente"
}
```

Código HTTP: 404 Not Found si la película no existe.

Eliminar películas:
Descripción: Permite que un administrador elimine una película.
Request:

Headers: Authorization: Bearer {access_token}
Endpoint: DELETE /api/v1/admin/movies/{id}

Response:

Código HTTP: 200 OK

```
{
  "message": "Película eliminada exitosamente"
}
```

Código HTTP: 404 Not Found si la película no existe.

Buscar películas (sin login):
Descripción: Permite buscar películas recomendadas, con filtros por título, actores y categorías.
Request:

Endpoint: GET /api/v1/movies?titulo={titulo}&actor={actor}&categoria={categoria}

[Aclaración: Este es un diseño básico del endpoint, también se podrían realizar filtros separados.]
Response:

Código HTTP: 200 OK

```
[
  {
    "id": 1,
    "título": "Película Ejemplo",
    "año": 2024,
    "duración": 120,
    "categoría": "Acción"
  }
]
```

Código HTTP: 404 Not Found si la película no existe.

Calificar películas (usuario logueado):
Descripción: Permite que un usuario califique una película con una puntuación del 1 al 10.
Request:

Headers: Authorization: Bearer {access_token}
Body:

```
{
  "rating": 6
}
```

Endpoint: POST /api/v1/movies/{id}/rate

Response:

Código HTTP: 200 OK

```
{
 "message": "Película calificada exitosamente"
}
```

Código HTTP: 404 Not Found si la película no existe.

Buscar otros usuarios, seguirlos y ver sus calificaciones (usuario logueado):
Descripción: Permite buscar otros usuarios, seguirlos y ver las películas que han calificado.
Request:

Headers: Authorization: Bearer {access_token}
Endpoint: POST /api/v1/users/{id}/follow

Response:

Código HTTP: 200 OK

```
{
 "message": "Siguiendo usuario"
}
```

Código HTTP: 404 Not Found si el usuario no existe.

Request:

Headers: Authorization: Bearer {access_token}
Endpoint: GET /api/v1/users/{id}/followers
Endpoint: GET /api/v1/users/{id}/following

Response:

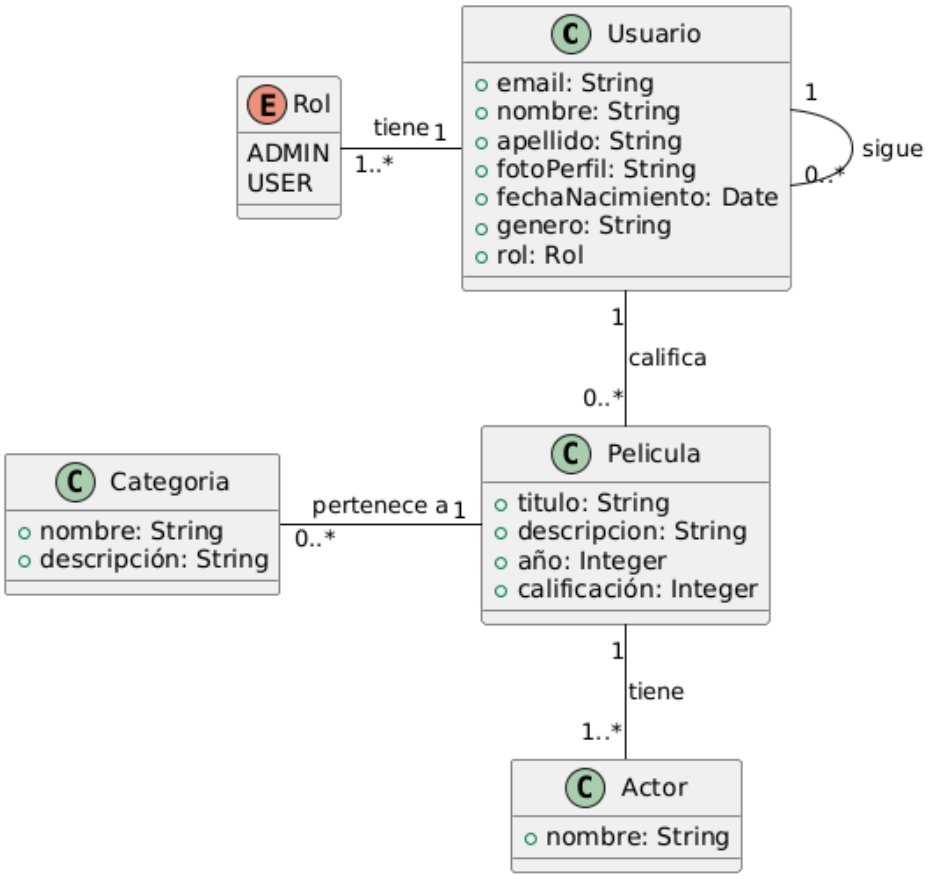
Código HTTP: 200 OK
{
"seguidores": ["Usuario1", "Usuario2"],
"seguidos": ["Usuario3", "Usuario4"]
}
Código HTTP: 404 Not Found si el usuario no existe.

Aceptar/rechazar solicitudes de seguimiento (usuario logueado):
Descripción: Permite aceptar o rechazar solicitudes de seguimiento.
Request:
Headers: Authorization: Bearer {access_token}
Body:
{
"status": "aceptado"
}
Endpoint: POST /api/v1/users/{id}/follow/request
Response:
Código HTTP: 200 OK
{
"message": "Solicitud aceptada"
}
Código HTTP: 404 Not Found si el usuario no existe.

Ver y editar perfil de usuario (usuario logueado):
Descripción: Permite ver y editar el perfil de usuario, con opción de eliminar cuenta.
Request:
Headers: Authorization: Bearer {access_token}
Body:
{
"nombre": "NuevoNombre",
"apellido": "NuevoApellido",
"género": "Masculino",
"avatar": "nueva_profilepic"
}
Endpoint: PATCH /api/v1/users/profile
Endpoint: GET /api/v1/users/profile
Response:
Código HTTP: 200 OK
{
"message": "Perfil actualizado"
}
Código HTTP: 400 Bad Request si hay error en la validación.

2. Modelo de dominio preliminar:

Es una versión compacta del dominio de la aplicación.



3. Propuesta de arquitectura preliminar:

Decido utilizar el patrón de arquitectura layers/capas, dado que se trata de una aplicación que los usuarios pueden descargar y crear una cuenta, en donde se realizan consultas a una base de datos a través de una API donde es necesario que estos datos persistan.

Siguiendo el modelo de vistas 4+1 contamos con:
Vista Lógica: Se apoya en los requisitos funcionales.

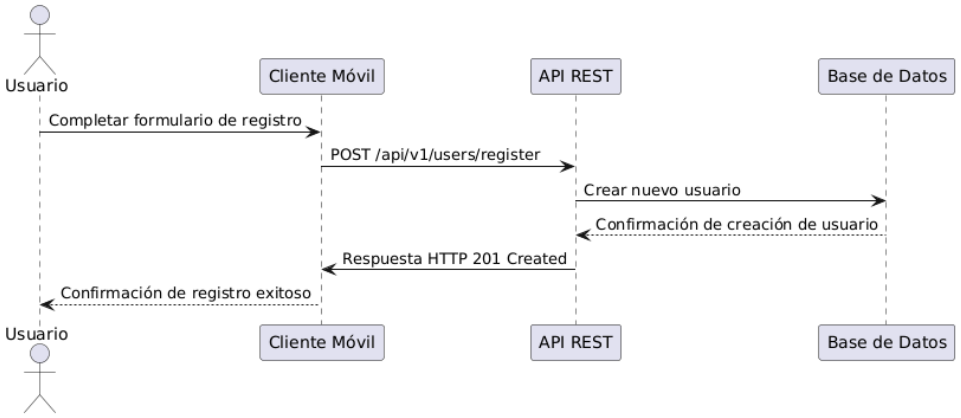
- Búsqueda y calificación de películas.
- Seguimiento de otros usuarios.
- Gestión de perfiles de usuario.

Vista de desarrollo (o componentes):
Descripción de los componentes
API REST: vendría a ser el Backend.

Base de datos: almacena y persiste los datos de la aplicación, como usuarios y películas con sus atributos cada uno.
Cliente Móvil: Interfaz de usuario que permite la interacción con la aplicación.
Servicio de autenticación: Está relacionado a la seguridad y a la autenticación/ autorización.

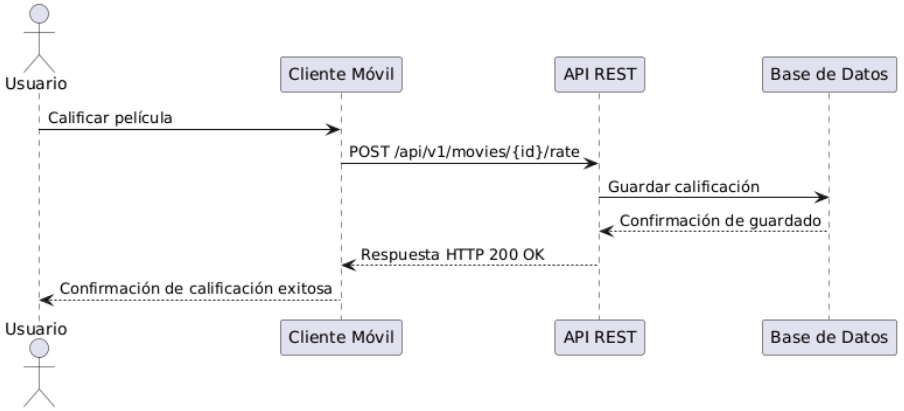
Vista de procesos:
Gráfico un posible flujo de trabajo en 2 procesos a modo de ejemplo.
Proceso de Registro:

1. El usuario envía sus datos a la API REST.
2. La API valida los datos y crea un nuevo registro en la base de datos.

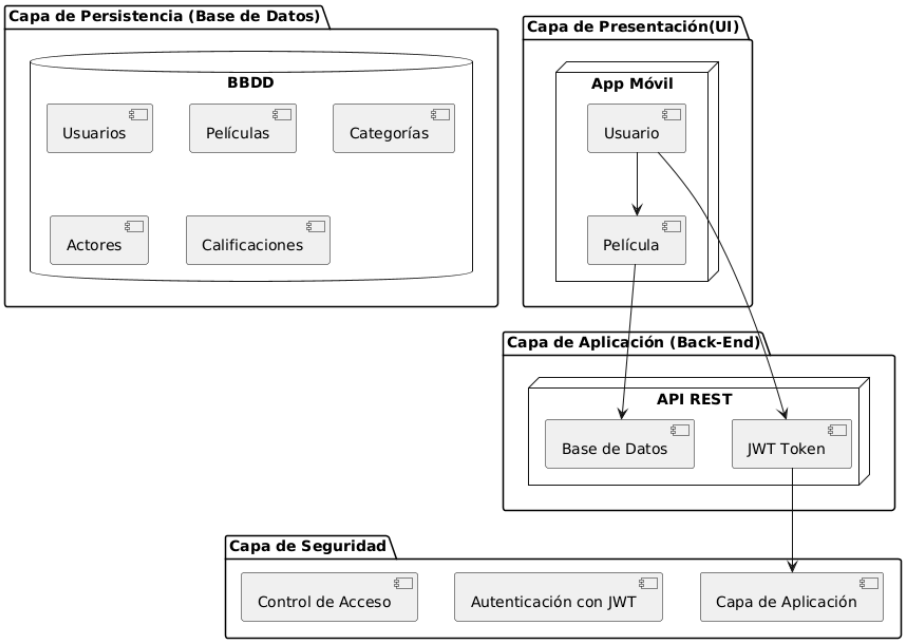


Proceso de Calificación:

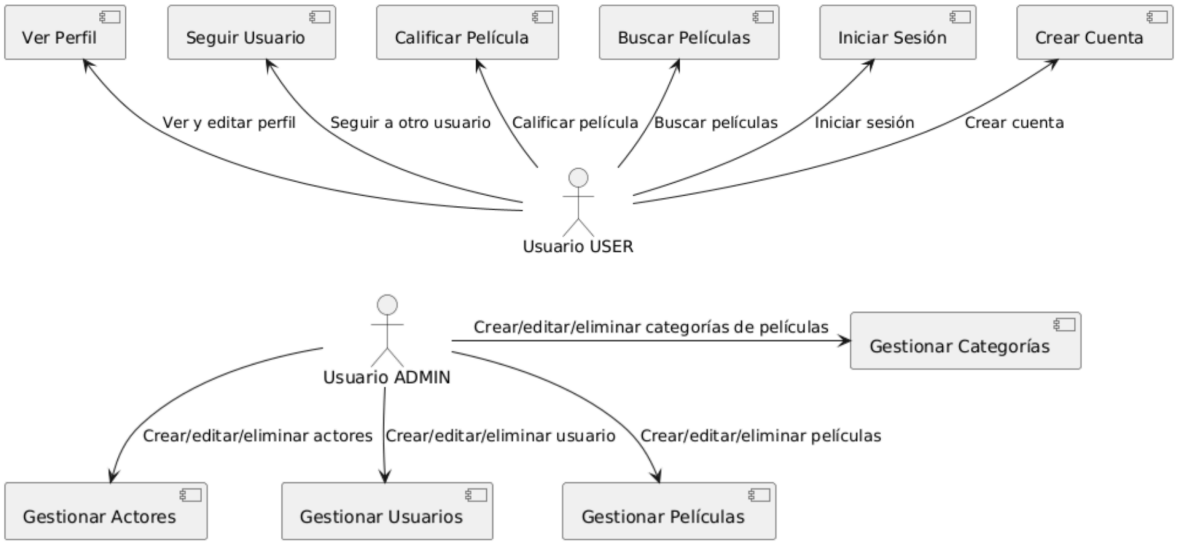
1. El usuario selecciona una película y envía su calificación.
2. La API actualiza la base de datos con la nueva calificación.



Vista FÍsica:
Muestro el modelo de capas.



Vista de Escenarios:
Presento los escenarios de uso de la aplicación, como crear cuenta, buscar películas y calificar estas mismas; dependiendo del rol que posee el usuario.



Comentarios de lo visto en clase en la revisión del ejercicio (23/09):
Faltaría agregar paginación en los requests donde se realizan request con GET en los que pueden llegar a traer muchos resultados.
En caso de que se haga un PUT esta bueno mandar en el body el json completo, en caso de querer modificar uno (o pocos) atributos es recomendable utilizar PATCH.