Documentación del Proyecto

1. Introducción

Este proyecto es una API backend desarrollada en Kotlin con Spring Boot. Su objetivo es gestionar la autenticación y el perfil de usuarios, permitiendo operaciones como registro, login, logout y actualización de datos.

2. Estructura del Proyecto

El proyecto está organizado en varias carpetas y archivos. A continuación, se explica cada una de ellas:

2.1. `src/main/kotlin/com/aperture_science/city_lens_api/`

Es la raíz del proyecto. Aquí se encuentran los paquetes principales que organizan la lógica de la aplicación.

2.2. `user/`

Contiene todo lo relacionado con la gestión de usuarios.

2.2.1. `controller/`

- `UsuarioController.kt`:
  - Es el controlador que maneja las solicitudes HTTP relacionadas con los usuarios.
  - Expone endpoints como `/v1/users/login`, `/v1/users/register`, `/v1/users/logout`, etc.
  - Se comunica con el servicio (`UsuarioService`) para procesar las solicitudes.

2.2.2. `controller/body/`

- `UsuarioLoginBody.kt`:
  - Representa los datos que un usuario envía al hacer login (correo y contraseña).

- `UsuarioLoginOutputBody.kt`:
  - Representa la respuesta después de un login exitoso (token y datos del usuario).
- `UsuarioPutMeBody.kt`:
  - Representa los datos que un usuario puede actualizar en su perfil (nombre, correo, contraseña).

2.2.3. `repository/`

- `UsuarioRepository.kt`:
  - Es el repositorio que interactúa directamente con la base de datos.
  - Contiene métodos para guardar, buscar, actualizar y eliminar usuarios y tokens de sesión.

2.2.4. `repository/entity/`

- `Usuario.kt`:
  - Representa la entidad `Usuario` en la base de datos.
  - Contiene campos como `id`, `email`, `firstName`, `lastName`, `password`, etc.
- `SessionToken.kt`:
  - Representa la entidad `SessionToken` en la base de datos.
  - Almacena tokens de sesión asociados a usuarios.

2.2.5. `service/`

- `UsuarioService.kt`:
  - Es el servicio que implementa la lógica de negocio.
  - Se encarga de validar credenciales, generar tokens, actualizar perfiles, etc.
  - Se comunica con el repositorio (`UsuarioRepository`) para acceder a los datos.

2.3. `util/`

Contiene utilidades generales que son usadas en diferentes partes del proyecto.

2.3.1. `EntityManagerFactoryInstance.kt`

- **Propósito**:
  - Es un singleton estático que permite compartir una instancia de `EntityManagerFactory` entre diferentes componentes del proyecto.
  - Facilita el acceso a la base de datos a través de `EntityManager`.
- **Uso**:
  - Se inicializa en la función `main` de la aplicación (`CityLensApiApplication.kt`).
  - Proporciona una única instancia de `EntityManagerFactory` para toda la aplicación.

2.3.2. `HashUtil.kt`

- **Propósito**:
  - Proporciona funciones para generar hashes SHA-256.
  - Se utiliza para hashear contraseñas y tokens de sesión.
- **Métodos**:
  - `generateHash(value: String)`: Genera un hash SHA-256 a partir de un string.
  - `hash(value: String)`: Función pública que llama a `generateHash` para obtener el hash.

2.4. `sql/`

Contiene los archivos SQL para configurar la base de datos.

2.4.1. `schema.sql`

- **Propósito**:
  - Define el esquema de la base de datos, incluyendo las tablas y relaciones.
- **Tablas principales**:
  - `Users`: Almacena la información de los usuarios (nombre, correo, contraseña, rol, etc.).
  - `Location`: Almacena las ubicaciones de los reportes.
  - `Image`: Almacena las URLs de las imágenes asociadas a los reportes.
  - `Report`: Almacena los reportes generados por los usuarios.
  - `Notification`: Almacena las notificaciones enviadas a los usuarios.

- `Moderation`: Almacena la información de moderación de los reportes.
- `Token`: Almacena los tokens de sesión de los usuarios.

- **Funciones y triggers**:
  - `enforce_moderator_role()`: Función que verifica si un usuario tiene permisos de moderador o administrador antes de permitir la moderación de un reporte.
  - `check_moderator_before_insert`: Trigger que ejecuta la función `enforce_moderator_role` antes de insertar un registro en la tabla `Moderation`.

2.4.2. `setup.sql`

- **Propósito**:
  - Contiene scripts adicionales para configurar la base de datos, como la creación de usuarios, roles o datos iniciales.
  - Puede incluir inserciones de datos de prueba o configuraciones específicas del entorno.

2.5. `CityLensApiApplication.kt`

- **Ubicación**: `src/main/kotlin/com/aperture_science/city_lens_api/`
- **Propósito**:
  - Es la clase principal de la aplicación Spring Boot.
  - Inicializa la aplicación y configura el `EntityManagerFactory`.
- **Funcionamiento**:
  - En la función `main`, se crea una instancia de `EntityManagerFactory` y se asigna a `EntityManagerFactoryInstance`.
  - Luego, se inicia la aplicación Spring Boot.

2.6. `application.properties`

- **Ubicación**: `src/main/resources/`
- **Propósito**:
  - Contiene las configuraciones de la aplicación, como la conexión a la base de datos y las propiedades de Swagger (OpenAPI).
- **Configuraciones clave**:
  - `spring.jpa.database-platform`: Define el dialecto de la base de datos (PostgreSQL).

- `spring.jpa.hibernate.ddl-auto`: Configura el comportamiento de Hibernate para actualizar el esquema de la base de datos.
- `spring.datasource.driver-class-name`: Especifica el driver de la base de datos (PostgreSQL).
- Configuraciones de Swagger para habilitar la interfaz de documentación de la API.

3. Flujo de Configuración de la Base de Datos

1. **Esquema de la Base de Datos**:
   - El archivo `schema.sql` define la estructura de la base de datos (tablas, relaciones, restricciones, etc.).
   - Se ejecuta automáticamente al iniciar la aplicación si `spring.jpa.hibernate.ddl-auto` está configurado como `update`.
2. **Configuración de la Aplicación**:
   - El archivo `application.properties` define cómo la aplicación se conecta a la base de datos y configura otras propiedades como Swagger.
3. **Inicialización de la Aplicación**:
   - En `CityLensApiApplication.kt`, se inicializa el `EntityManagerFactory` para permitir el acceso a la base de datos.

4. Flujo del Proyecto

1. **Registro de Usuario**:
   - El frontend envía los datos del usuario a `/v1/users/register`.
   - El controlador (`UsuarioController`) recibe los datos y los pasa al servicio (`UsuarioService`).
   - El servicio crea un nuevo usuario y lo guarda en la base de datos usando el repositorio (`UsuarioRepository`).
2. **Login de Usuario**:
   - El frontend envía el correo y la contraseña a `/v1/users/login`.
   - El controlador recibe los datos y los pasa al servicio.
   - El servicio valida las credenciales y genera un token de sesión si son correctas.
   - El token y los datos del usuario se devuelven al frontend.
3. **Logout de Usuario**:

- El frontend envía el token de sesión a `/v1/users/logout`.
- El controlador recibe el token y lo pasa al servicio.
- El servicio elimina el token de la base de datos.

4. **Actualización de Perfil**:
   - El frontend envía los nuevos datos del usuario a `/v1/users/me`.
   - El controlador recibe los datos y los pasa al servicio.
   - El servicio actualiza los datos del usuario en la base de datos.

5. Ejemplo de Solicitudes

5.1. Registro

```
POST /v1/users/register
Content-Type: application/json

{
  "first_name": "Juan",
  "last_name": "P rez",
  "email": "juan@example.com",
  "password": "contrase a123"
}
```

5.2. Login

```
POST /v1/users/login
Content-Type: application/json

{
  "email": "juan@example.com",
  "password": "contrase a123"
}
```

5.3. Logout

```
POST /v1/users/logout
Authorization: hashed_token_abc123
```

5.4. Obtener Perfil

```
GET /v1/users/me
Authorization: hashed_token_abc123
```

5.5. Actualizar Perfil

```
PUT /v1/users/me
Authorization: hashed_token_abc123
Content-Type: application/json

{
  "first_name": "Ana",
  "email": "ana@example.com"
}
```

6. Ejecución del Proyecto

1. **Configura la Base de Datos**:
   - Asegúrate de que PostgreSQL esté instalado y configurado.
   - Crea una base de datos y actualiza las credenciales en `application.properties`.
2. **Inicia la Aplicación**:
   - Ejecuta la función `main` en `CityLensApiApplication.kt`.
   - La aplicación creará las tablas y relaciones definidas en `schema.sql`.
3. **Prueba la API**:
   - Usa Swagger (disponible en `/swagger-ui-custom.html`) para probar los endpoints de la API.