Documentación API Biblioteca

Daniel Hernández Gómez

Enlace a github: https://github.com/Danonexe/Proyecto-API-REST-Segura-Biblioteca

Índice:

1. Introducción del proyecto

Título proyecto:

Biblioteca Gómez

Idea del proyecto:

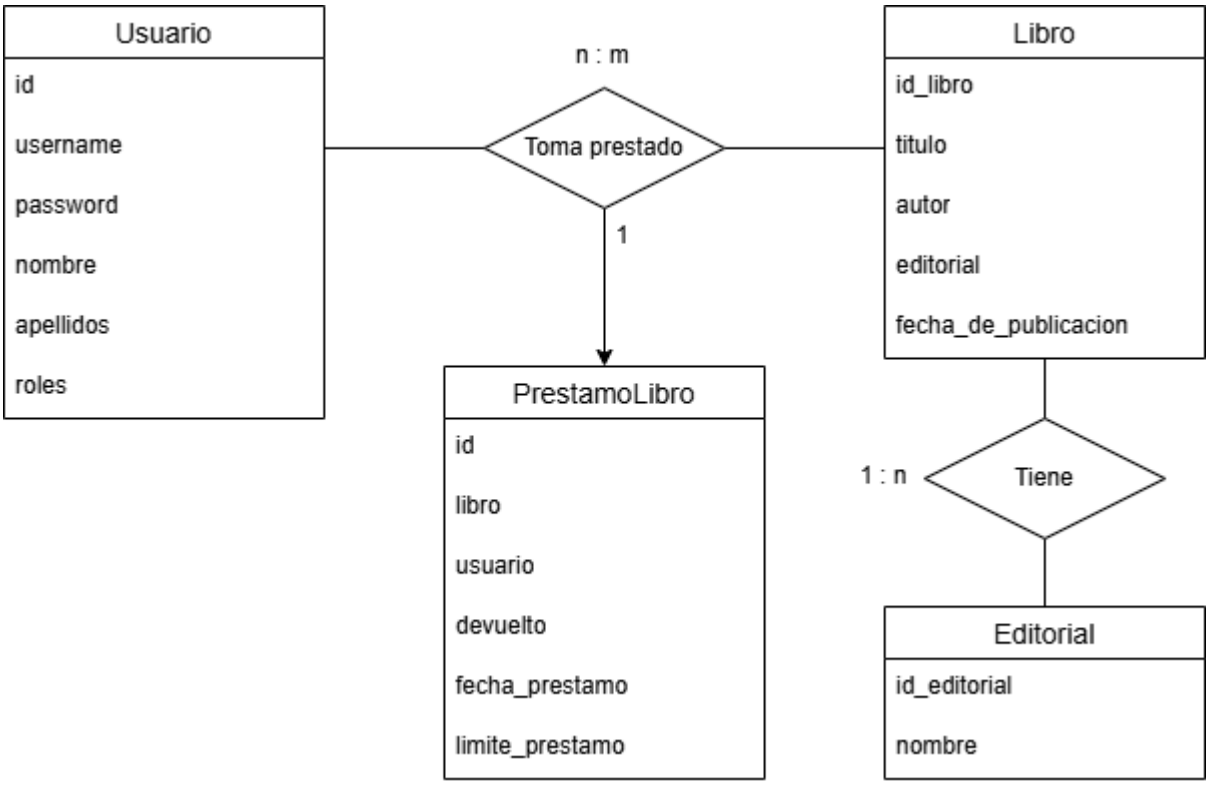
La idea del proyecto es crear una API que con una conexión a una base de datos gestione las el préstamo de libros y el registro de usuarios y libros.

Justificación del proyecto

Necesitamos una aplicación para controlar todas las gestiones de usuarios, libros y préstamos, mejorando la eficiencia y experiencia tanto del cliente como los empleados que la usen.

2. Descripción de las tablas

El sistema está construido alrededor de cuatro entidades principales: Usuario, Editorial, Libro, y PrestamoLibro.



Usuarios

Representa a los usuarios registrados en el sistema.

- **id** (Primary Key): Identificador único del usuario.
- **username** (Unique, Not Null): Nombre de usuario.
- **password** (Not Null): Contraseña cifrada.
- **nombre** (Not Null): Nombre del usuario.
- **apellidos** (Not Null): Apellidos del usuario.
- **roles** (Not Null): Roles asignados al usuario.

Data class de la API:

```
@Entity
@Table(name = "usuarios")
data class Usuario(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id: Long? = null,

    @Column(unique = true, nullable = false)
    var username: String? = null,

    @Column(nullable = false)
    var password: String? = null,

    @Column(nullable = false)
    var nombre: String? = null,

    @Column(nullable = false)
    var apellidos: String? = null,

    @Column(nullable = false)
    var roles: String? = null

)
```

Editoriales

Representa a las editoriales responsables de la publicación de libros.

- **id_editorial** (Primary Key): Identificador único de la editorial.
- **nombre** (Not Null): Nombre de la editorial.

Data class de la API:

```
@Entity
@Table(name = "editoriales")
data class Editorial(
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id_editorial: Long? = null,

    @Column(nullable = false)
    var nombre: String? = null,

)
```

Libros

Representa a los libros disponibles en el sistema.

- **id_libro** (Primary Key): Identificador único del libro.
- **titulo** (Not Null): Título del libro.
- **autor** (Not Null): Autor del libro.
- **editorial** (Foreign Key): Relación con la tabla editoriales.
- **fecha_de_publicacion** (Not Null): Fecha de publicación del libro.

Data class de la API:

```
@Entity
@Table(name = "libros")
data class Libro(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id_libro: Long? = null,

    @Column(nullable = false)
    var titulo: String? = null,

    @Column(nullable = false)
    var autor: String? = null,

    @ManyToOne
    @JoinColumn(name = "id_editorial", nullable = false)
    var editorial: Editorial? = null,

    @Column(nullable = false)
    var fecha_de_publicacion: Date? = null,

)
```

PrestamoLibro

Representa el préstamo de un libro a un usuario.

- **id** (Primary Key): Identificador único del préstamo.
- **libro** (Foreign Key): Relación con la tabla libros.
- **usuario** (Foreign Key): Relación con la tabla usuarios.
- **devuelto:** Indica si el libro ha sido devuelto.
- **fecha_prestamo**: Fecha en la que se realizó el préstamo.
- **limite_prestamo**: Fecha límite para devolver el libro.

Data class de la API:

```
@Entity
@Table(name = "libros")
data class Libro(
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    var id_libro: Long? = null,

    @Column(nullable = false)
    var titulo: String? = null,

    @Column(nullable = false)
    var autor: String? = null,

    @ManyToOne
    @JoinColumn(name = "id_editorial", nullable = false)
    var editorial: Editorial? = null,

    @Column(nullable = false)
    var fecha_de_publicacion: Date? = null,

)
```

3. Endpoints

Un endpoint en una API es una dirección específica que actúa como punto de acceso para interactuar con los recursos de la API. A continuación voy a explicar todos los endpoints que tiene mi código.

Usuarios

- **POST /usuarios/login**
  Permite iniciar sesión en el sistema.
- **POST /usuarios/register**
  Permite registrar un nuevo usuario en el sistema.
- **GET /usuarios**
  Obtiene la lista de todos los usuarios (requiere rol ADMIN).

- **GET /usuarios/{id}**
  Obtiene la información de un usuario específico por su ID (requiere rol ADMIN).
- **PUT /usuarios/{id}**
  Actualiza la información de un usuario específico por su ID (requiere rol ADMIN).
- **DELETE /usuarios/{id}**
  Elimina un usuario específico por su ID (requiere rol ADMIN).

Editoriales

- **POST /editoriales/create**
  Permite crear una nueva editorial (requiere rol ADMIN).
- **GET /editoriales**
  Obtiene la lista de todas las editoriales (requiere autenticación).
- **GET /editoriales/{nombre}**
  Obtiene la información de una editorial específica por su nombre (requiere autenticación).
- **PUT /editoriales/{nombre}**
  Actualiza la información de una editorial específica por su nombre (requiere rol ADMIN).
- **DELETE /editoriales/{nombre}**
  Elimina una editorial específica por su nombre (requiere rol ADMIN).
- **GET /editoriales/{nombreEditorial}/libros**
  Obtiene la lista de libros de una editorial específica (requiere autenticación).

Libros

- **POST /libros/create**
  Permite crear un nuevo libro (requiere rol ADMIN).
- **GET /libros**
  Obtiene la lista de todos los libros (requiere autenticación).
- **GET /libros/{id}**
  Obtiene la información de un libro específico por su ID (requiere autenticación).
- **PUT /libros/{id}**
  Actualiza la información de un libro específico por su ID (requiere rol ADMIN).
- **DELETE /libros/{id}**
  Elimina un libro específico por su ID (requiere rol ADMIN).

Préstamos

- **POST /prestamos/create**
  Permite crear un nuevo préstamo de libro (requiere rol ADMIN).
- **PUT /prestamos/devolver/{id}**
  Permite devolver un préstamo específico por su ID (requiere rol ADMIN).
- **GET /prestamos**
  Obtiene la lista de todos los préstamos (requiere rol ADMIN).
- **GET /prestamos/{id}**
  Obtiene la información de un préstamo específico por su ID (requiere rol ADMIN).

4. Lógica de negocio

Como biblioteca, queremos brindarles a nuestros usuarios una experiencia sencilla y organizada para acceder a los libros. Los usuarios podrán:

1. **Ver el catálogo de libros disponibles**: Esto significa que tendrán acceso a una lista o sistema donde puedan explorar los libros, ver detalles como título, autor, género, y disponibilidad.
2. **Solicitar un préstamo**: Si están interesados en un libro, podrán informar a los encargados de la biblioteca para que lo marquen como "prestado". Esto asegura que se registre oficialmente quién tiene el libro, evitando confusiones.

Por otro lado, los **encargados de la biblioteca** tendrán la responsabilidad de:

1. **Gestionar los registros de libros, editoriales y usuarios** (actualizar información, agregar nuevos libros, registrar las editoriales relacionadas).
2. **Registrar los préstamos** en el sistema cuando un usuario tome un libro, manteniendo así un control claro y actualizado del inventario.

También hemos controlado por código una serie de reglas para ayudar a esta lógica de negocio.

- Ninguno de los campos con relevancia para la información pueden ser nulos, como el usuario, el título de un libro etc
- A la hora de registrar un libro, un libro puede tener el mismo nombre que otro, pero si también tienen la misma editorial, la operación será declarada como incorrecta ya que solo queremos una copia de un libro, pero podemos tener varias ediciones diferentes.
- A la hora de crear un préstamo comprueba si ese libro que está siendo prestado ya se encuentra siendo prestado.
- Para que un usuario pueda hacer un préstamo, no puede tener otro sin devolver.
- A la hora de crear un préstamo se guarda la fecha del préstamo (la actual) y la de el plazo máximo para devolverlo.
- Las contraseñas de los usuarios han de ser mínimo de 8 caracteres

5. Excepciones y Códigos de Estado

En mi programa, manejo situaciones específicas utilizando excepciones personalizadas que se asocian a códigos de estado HTTP. Esto permite una comunicación clara sobre los problemas que puedan surgir al interactuar con el sistema. Las excepciones y sus correspondientes códigos de estado son las siguientes:

ValidationException - 400 Bad Request

Esta excepción se utiliza cuando el usuario envía datos inválidos o mal formados en una solicitud. El código de estado HTTP 400 indica que el problema está relacionado con el contenido enviado por el cliente, y se puede usar para errores como:

- Campos requeridos que están vacíos.
- Formatos incorrectos (por ejemplo, una contraseña no válida).

ConflictException - 409 Conflict

Esta excepción se lanza cuando hay un conflicto con el estado actual del recurso en el servidor. Se usa en situaciones donde la solicitud no puede completarse debido a reglas de negocio o restricciones de integridad.

Ejemplos:

- Intentar registrar un libro con el mismo título y editorial que otro ya existente.
- Crear una editorial con un nombre que ya está registrado en la base de datos.
-

NotFoundException - 404 Not Found

Esta excepción se lanza cuando el recurso solicitado por el usuario no existe o no puede ser encontrado. El código 404 indica que la URL o el identificador proporcionado en la solicitud no corresponde a ningún recurso disponible en el sistema.
Ejemplos:

- Intentar acceder a un libro que no está registrado en la biblioteca.
- Buscar un usuario o editorial que no exista en la base de datos.

6. Restricciones de seguridad

El programa tiene un control mediante autenticación para controlar qué funciones tiene cada usuario.

Usuarios no autenticados:

Solo podrán acceder a las siguientes rutas:

- /usuarios/login (método POST): Inicio de sesión.
- /usuarios/register (método POST): Registro.

Usuarios autenticados:

Si su rol es ADMIN:

- Podrá ver todos los usuarios: GET /usuarios.
- Podrá ver un usuario por ID: GET /usuarios/{id}.
- Podrá modificar un usuario por ID: PUT /usuarios/{id}.
- Podrá eliminar un usuario por ID: DELETE /usuarios/{id}.
- Podrá crear editoriales: POST /editoriales/create.
- Podrá ver todas las editoriales: GET /editoriales.
- Podrá ver una editorial específica: GET /editoriales/{nombre}.
- Podrá ver los libros asociados a una editorial: GET /editoriales/{nombreEditorial}/libros.
- Podrá modificar editoriales: PUT /editoriales/{nombre}.
- Podrá eliminar editoriales: DELETE /editoriales/{nombre}.
- Podrá crear libros: POST /libros/create.
- Podrá ver todos los libros: GET /libros.
- Podrá ver un libro por ID: GET /libros/{id}.
- Podrá modificar libros: PUT /libros/{id}.
- Podrá eliminar libros: DELETE /libros/{id}.
- Podrá crear préstamos: POST /prestamos/create.
- Podrá registrar la devolución de un préstamo: PUT /prestamos/devolver/{id}.
- Podrá ver todos los préstamos: GET /prestamos.
- Podrá ver un préstamo por ID: GET /prestamos/{id}.

Si su rol es USER:

- Podrá ver todas las editoriales: GET /editoriales.
- Podrá ver una editorial específica: GET /editoriales/{nombre}.
- Podrá ver los libros asociados a una editorial: GET /editoriales/{nombreEditorial}/libros.
- Podrá ver todos los libros: GET /libros.
- Podrá ver un libro por ID: GET /libros/{id}.

7. Preguntas

1. ¿Qué tecnologías has usado?

Para desarrollar el proyecto, utilicé la página web start.springboot.io para generar el paquete base, agregando dependencias como Spring Web, Spring Data JPA, OAuth2 Resource Server y Spring Boot DevTools.

El desarrollo se realizó en IntelliJ IDEA como IDE, facilitando la escritura del código. Para probar los endpoints, empleé Insomnia, aprovechando sus herramientas para pruebas

HTTP. Además, utilicé XAMPP para desplegar un servidor Apache y alojar la base de datos MySQL.

2. ¿Qué es una API REST? ¿Cuáles son los principios de una API REST? ¿Dónde identificas dichos principios dentro de tu implementación?

Una API REST es un sistema que permite la comunicación entre cliente y servidor a través de operaciones HTTP estándar como GET, POST, PUT y DELETE. Se centra en recursos, identificados por URL, y proporciona un diseño sencillo, escalable y flexible para integrar aplicaciones.

Los principios de una API REST son los siguientes:

1. **Sin estado (Stateless):** Cada solicitud del cliente al servidor es independiente, ya que no se guarda información sobre el estado entre las peticiones.
2. **Recursos identificados por URLs:** Los recursos (datos o entidades) se representan con identificadores únicos, usualmente en forma de URLs.
3. **Operaciones basadas en HTTP:**
    - **GET:** Recuperar datos de un recurso.
    - **POST:** Crear un recurso nuevo.
    - **PUT:** Modificar un recurso existente.
    - **DELETE:** Eliminar un recurso.
4. **Representación de recursos:** Los datos pueden enviarse en formatos como JSON (el más utilizado), XML o texto simple.
5. **Códigos de estado HTTP:** Las respuestas del servidor incluyen códigos estándar para indicar el resultado, como:
    - **200 OK:** Solicitud exitosa.
    - **404 Not Found:** Recurso no encontrado.
    - **500 Internal Server Error:** Error del servidor.
6. **Interfaz uniforme:** Se definen reglas claras para interactuar con los recursos, lo que hace que sea más sencillo de entender y usar.

En mi implementación, los principios de una API REST se reflejan así:

- **Sin estado:** Cada solicitud es independiente, como en /usuarios/login o /usuarios/register.
- **Recursos identificados por URLs:** Recursos claros, como /usuarios/{id}, /libros/{id} o /prestamos/{id}.
- **Operaciones HTTP:** Uso de GET, POST, PUT, DELETE para manejar recursos.
- **Representación de recursos**: Intercambio de datos en formato JSON.
- **Códigos de estado HTTP:** Uso de códigos estándar como 200 OK, 400 Bad Request, o 404 Not Found.
- **Interfaz uniforme:** Reglas consistentes para acceder y manipular recursos, simplificando el uso de la API.

3. ¿Qué ventajas tiene realizar una separación de responsabilidades entre cliente y servidor?

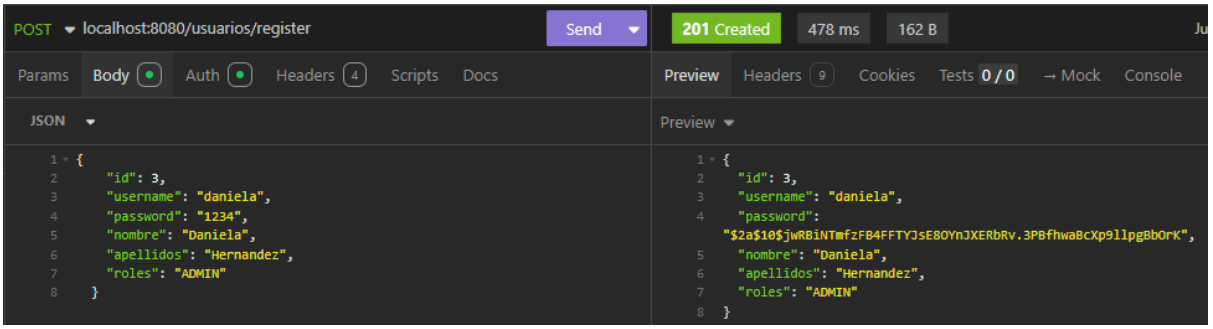
La división de responsabilidades entre cliente y servidor presenta varias ventajas importantes:

- **Escalabilidad:** Cada parte puede ampliarse de manera independiente según las demandas. Por ejemplo, es posible incrementar los recursos del servidor sin alterar el funcionamiento del cliente.
- **Mantenibilidad:** Simplifica las tareas de mantenimiento y actualización, ya que las modificaciones realizadas en la lógica del cliente no impactan directamente al servidor, y viceversa.
- **Reutilización:** El servidor puede ser compartido por distintos clientes al proporcionar servicios estándar a través de APIs.
- **Seguridad:** Al mantener la lógica más crítica y sensible en el servidor, se reduce su exposición al usuario final, disminuyendo el riesgo de manipulación directa.
- **Rendimiento:** El cliente puede encargarse de ciertas tareas locales, lo que aligera la carga del servidor y mejora la experiencia del usuario.
- **Flexibilidad:** Permite adoptar nuevas tecnologías en uno de los componentes sin generar un impacto directo en el otro.

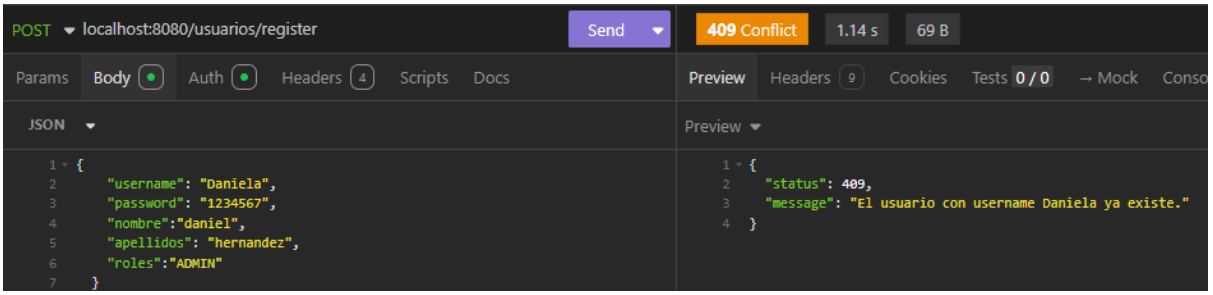
8. Pruebas

UsuarioController

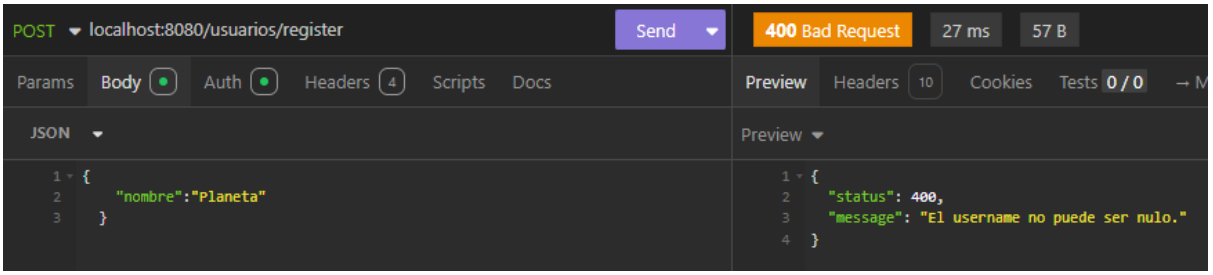
- **Endpoints**:
    - **POST /usuarios/register**: Registra un nuevo usuario.



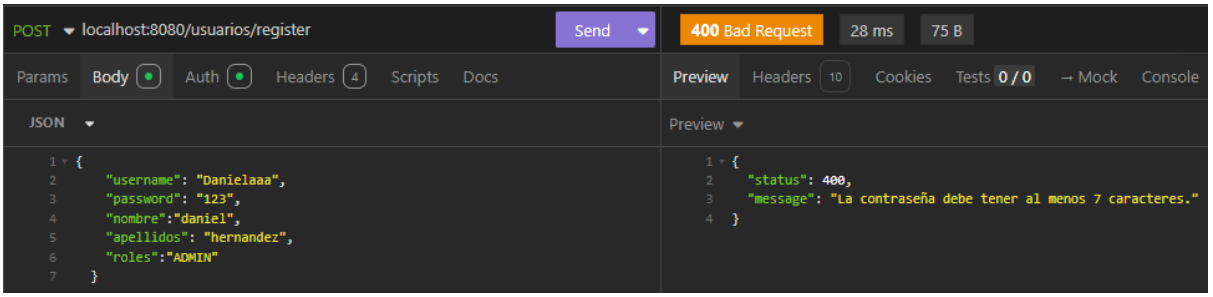
Si ya existe un usuario con ese username:



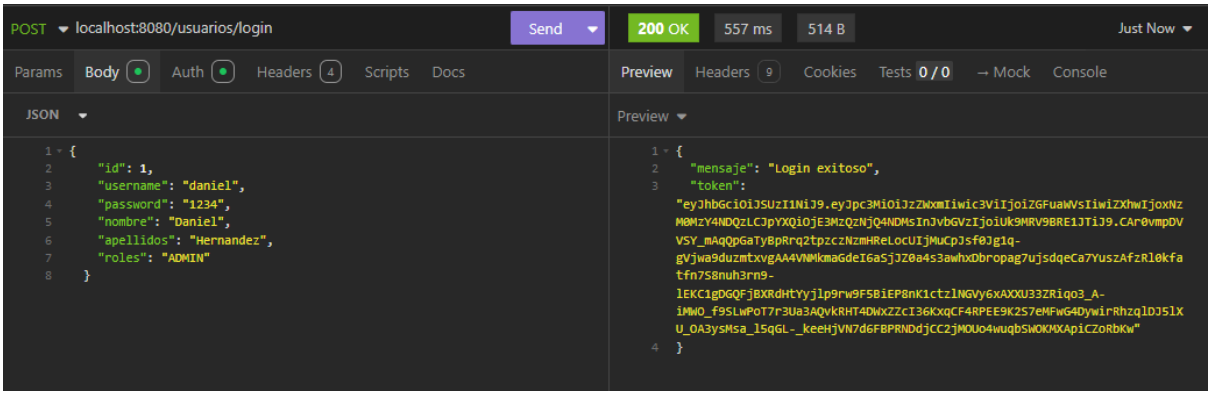
Si le faltan valores:



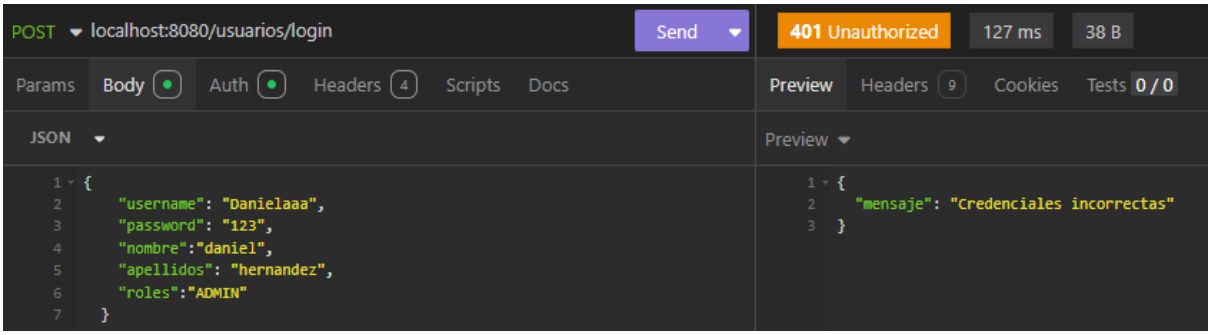
Si la contraseña no es apta:



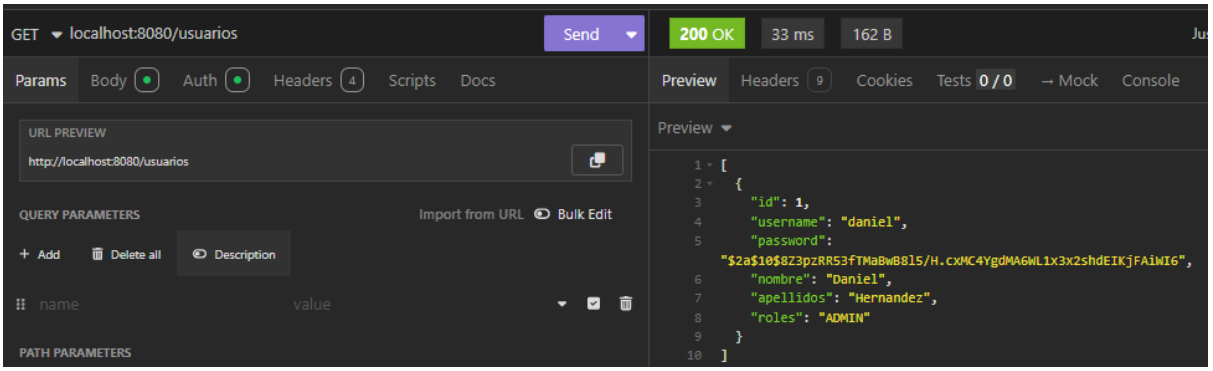
- **POST /usuarios/login**: Autentica un usuario y genera un token JWT.



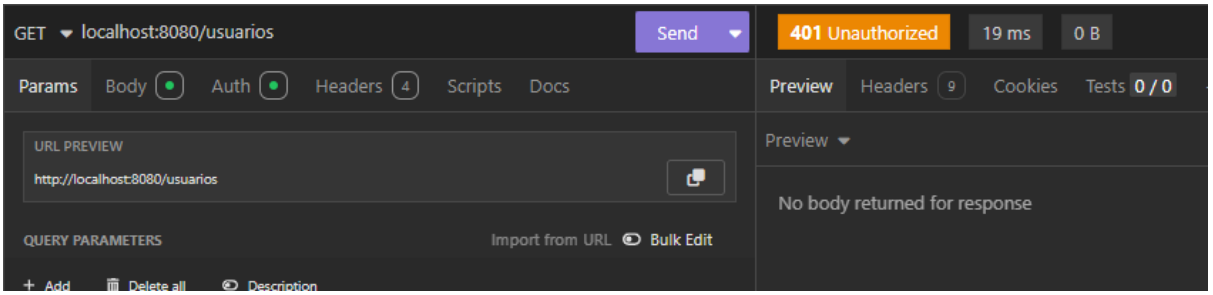
Si no existe el usuario o la contraseña es incorrecta:



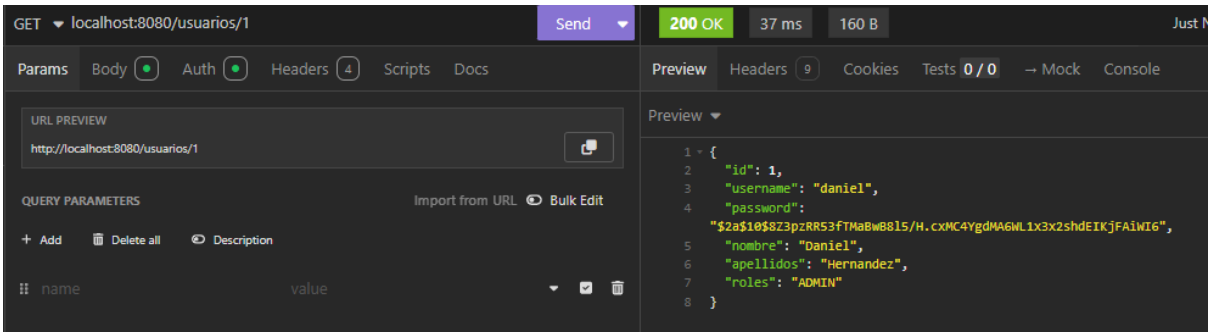
- **GET /usuarios**: Lista todos los usuarios.



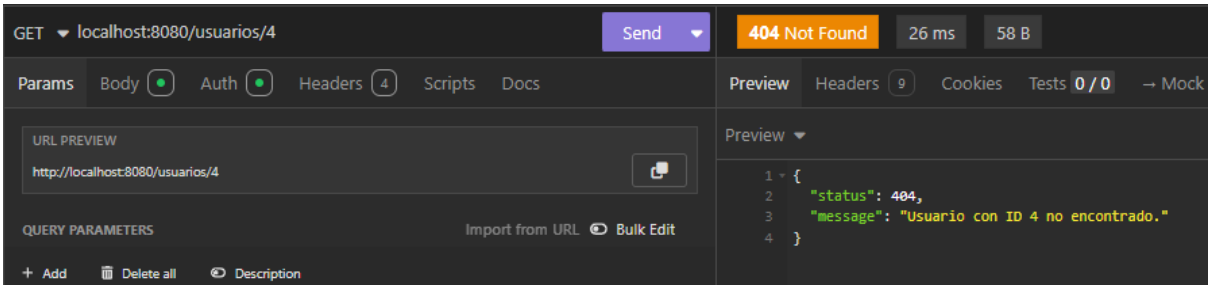
si no estás autorizado:



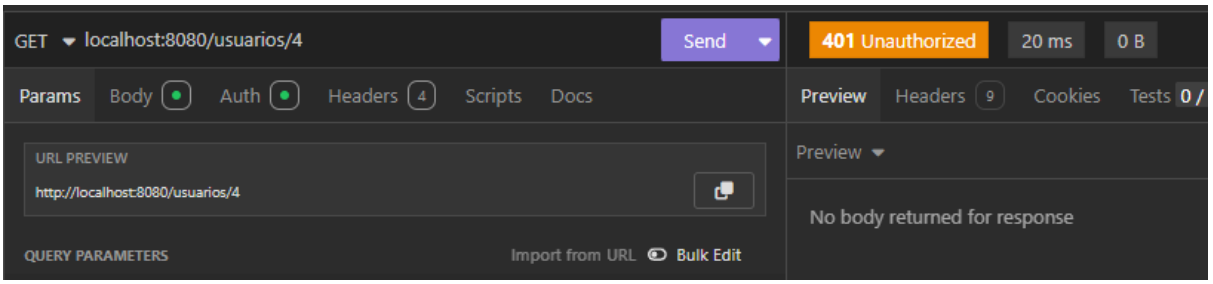
- **GET /usuarios/{id}**: Obtiene un usuario por su ID.



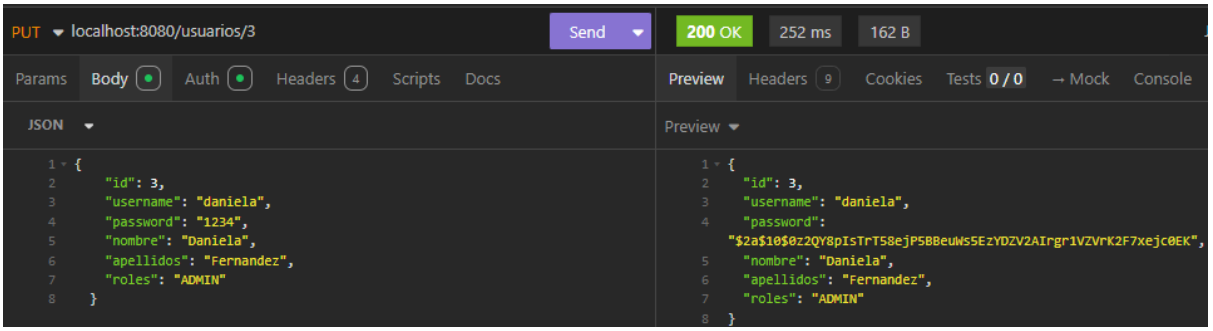
Si no existe el id que buscas



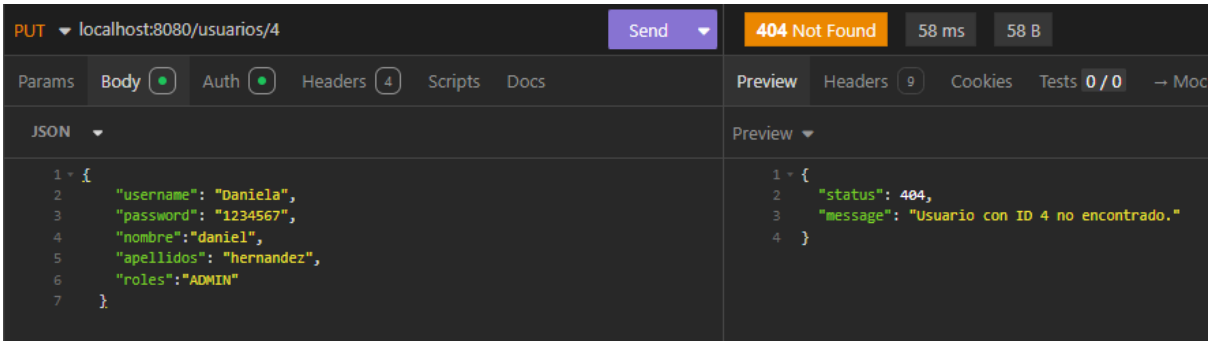
Si no tienes autorización (funciona en todos los lados donde no tienes autorización):



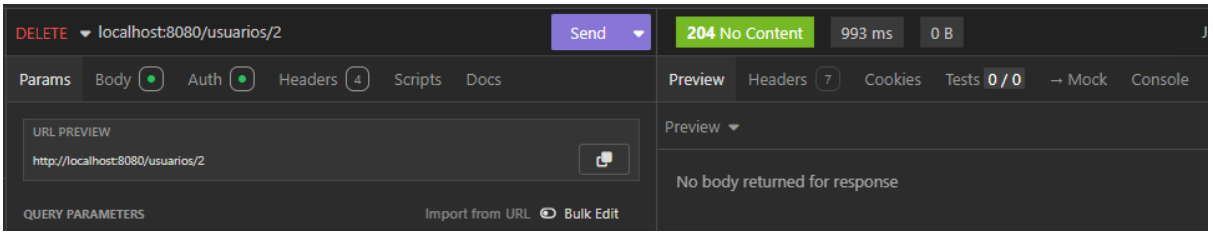
- **PUT /usuarios/{id}**: Actualiza un usuario.



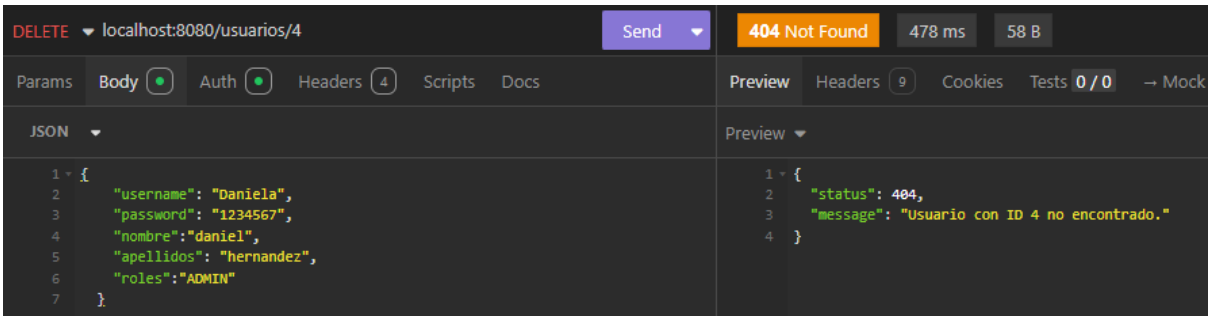
Si no existe ese Usuario



- **DELETE /usuarios/{id}**: Elimina un usuario.

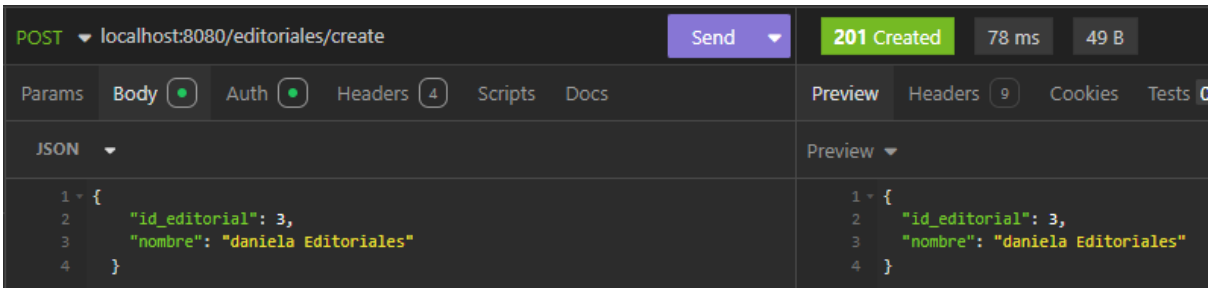


Si ese usuario no existe:

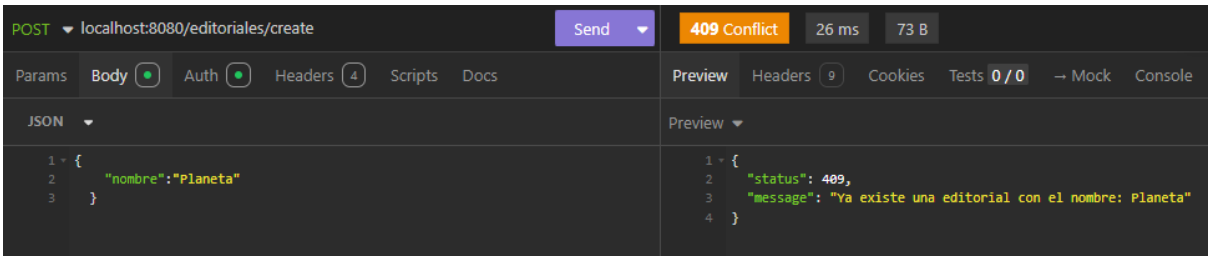


EditorialController

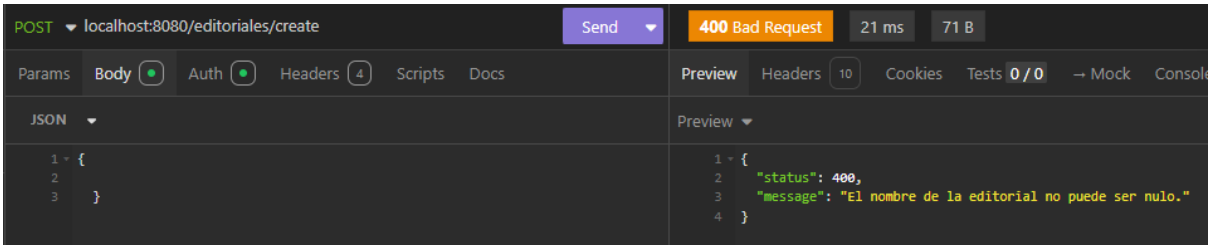
- **Endpoints**:
  - **POST /editoriales/create**: Crea una nueva editorial.



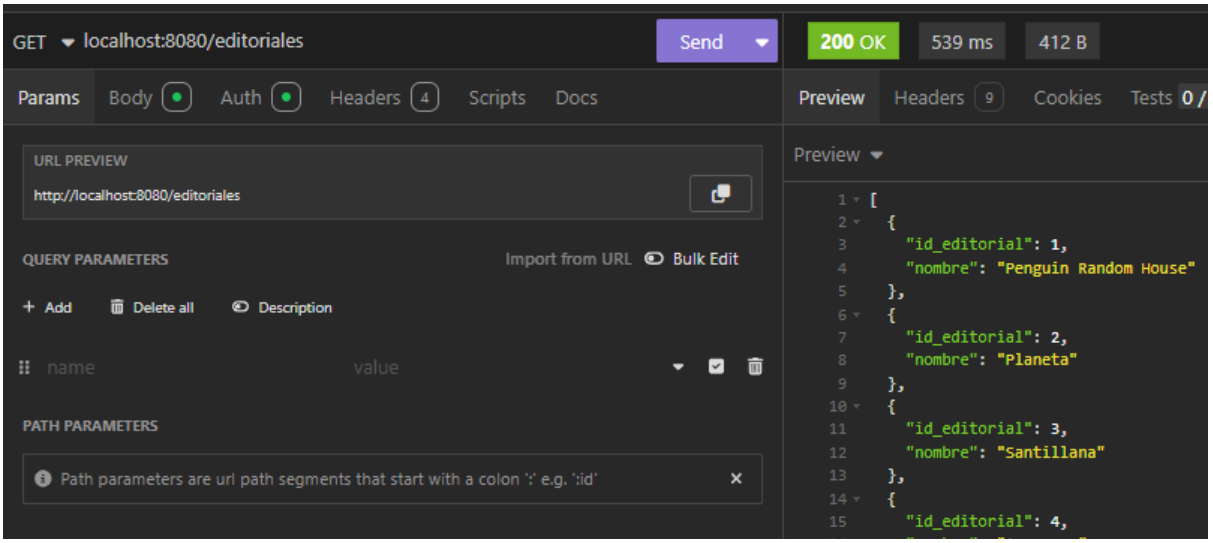
Si ya existe una editorial con ese nombre:



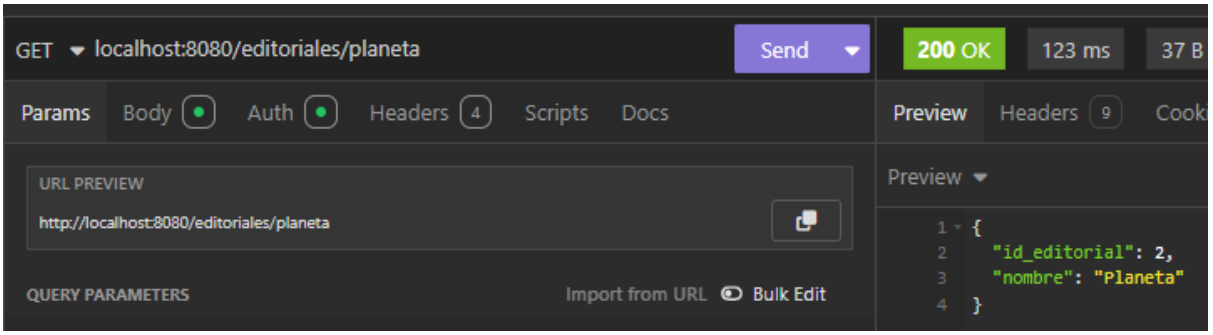
Si le faltan valores:



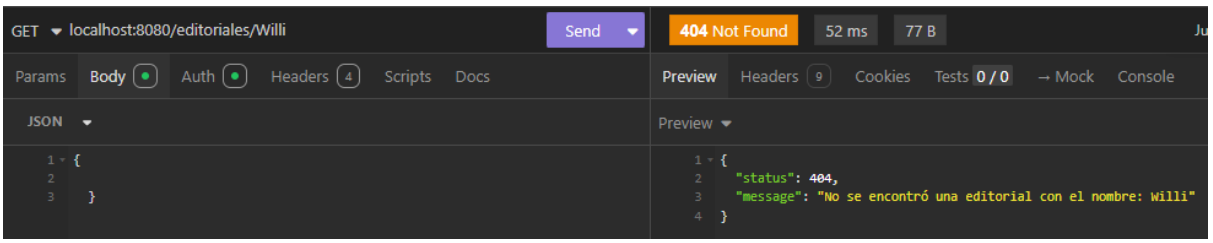
- **GET /editoriales**: Lista todas las editoriales



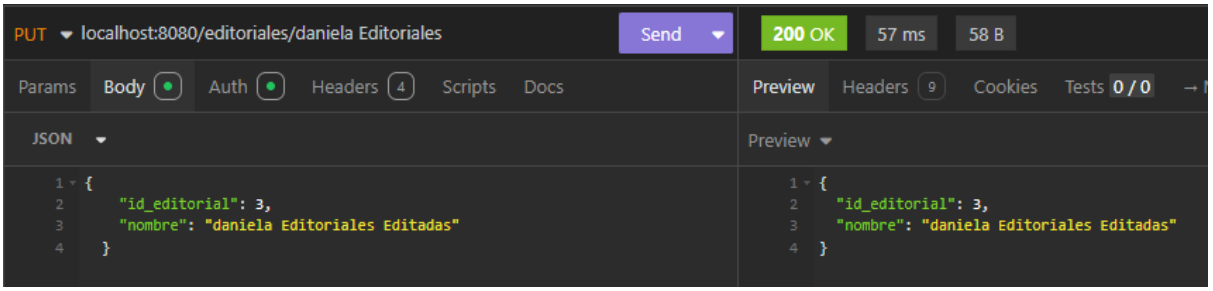
- **GET /editoriales/{nombre}**: Obtiene una editorial por su nombre.



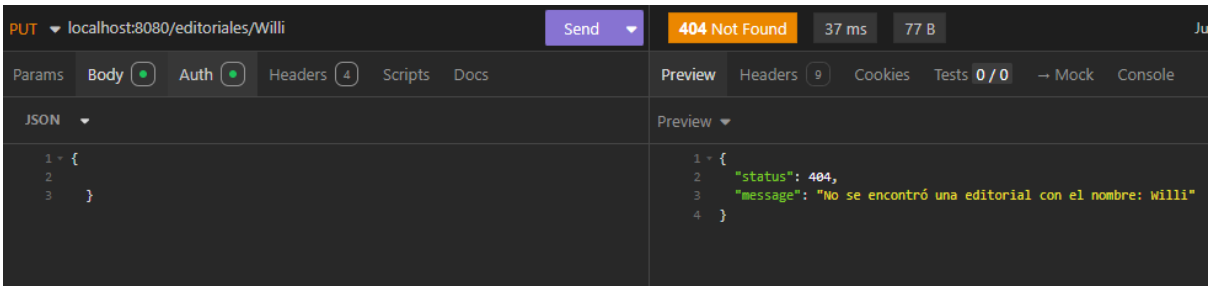
Si no existe ese nombre:



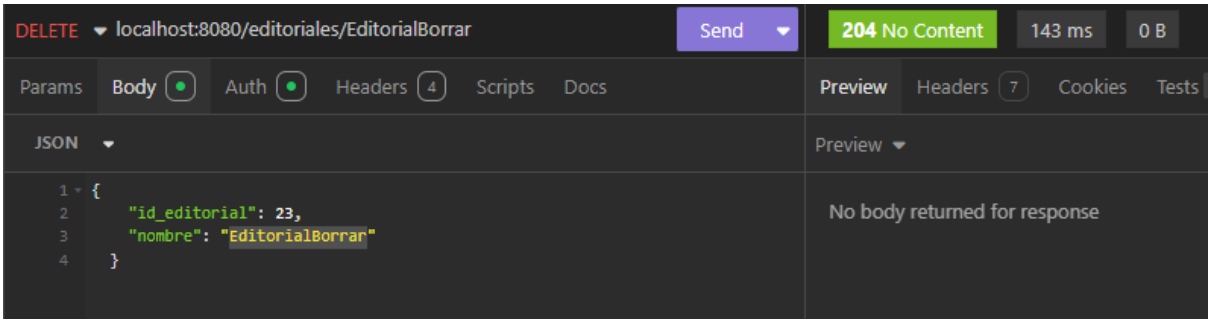
- **PUT /editoriales/{nombre}**: Actualiza una editorial.



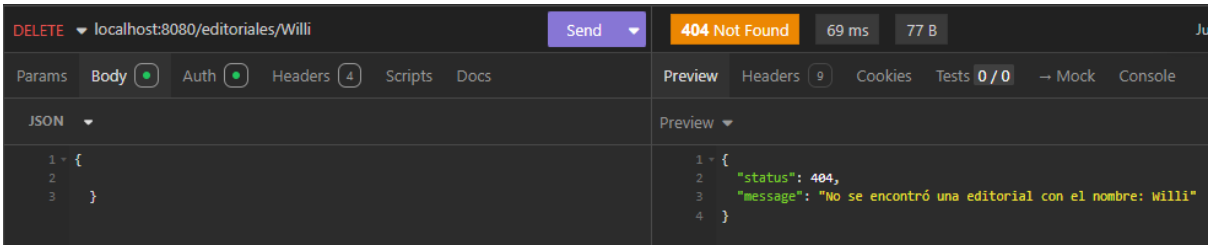
Si no existe ese nombre:



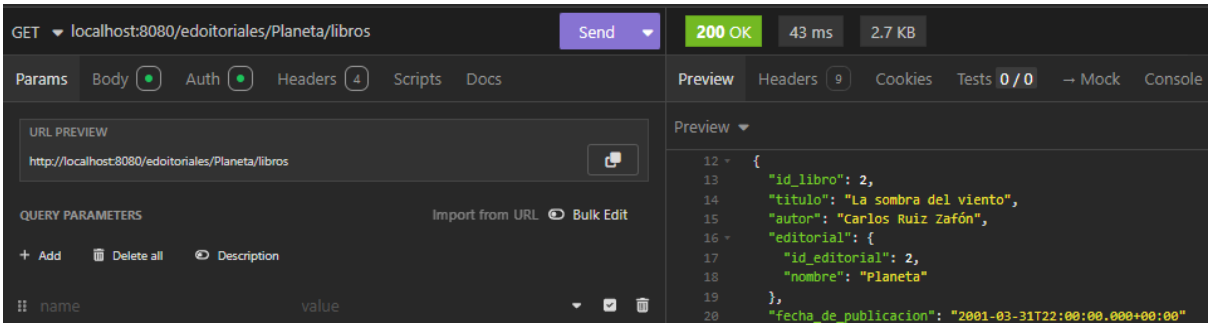
- **DELETE /editoriales/{nombre}**: Elimina una editorial.



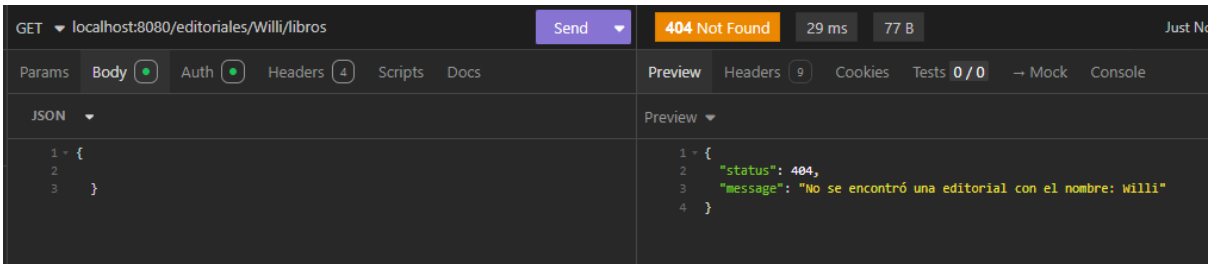
Si no existe ese nombre:



- **GET /editoriales/{nombreEditorial}/libros**: Lista los libros de una editorial.

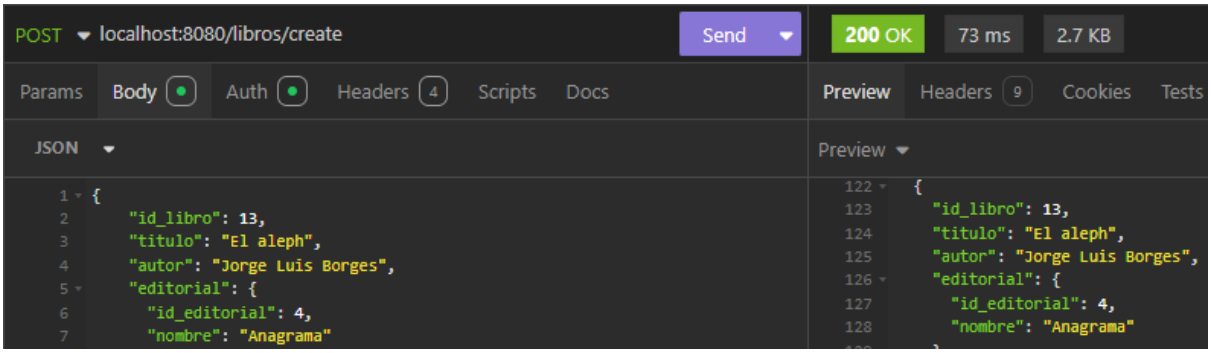


Si no existe ese nombre:

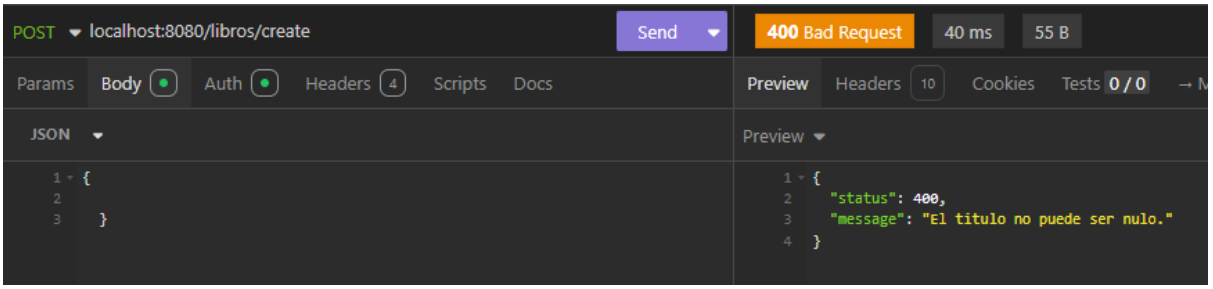


LibroController

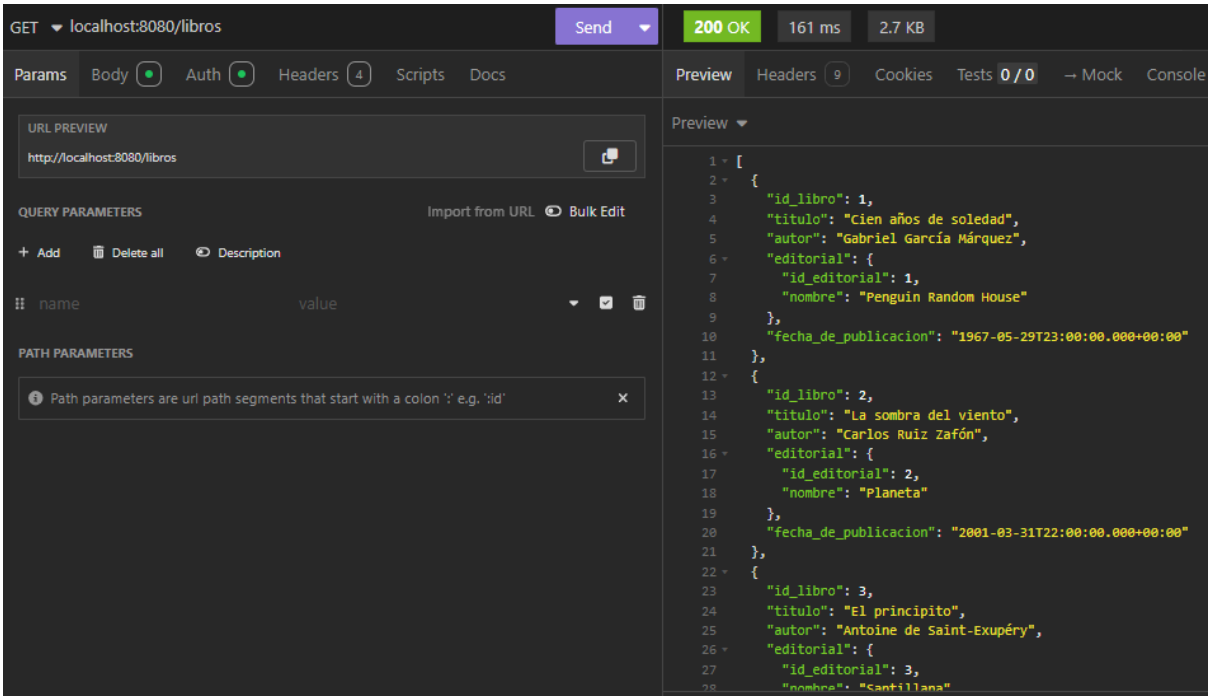
- **Endpoints**:
  - **POST /libros/create**: Crea un nuevo libro.



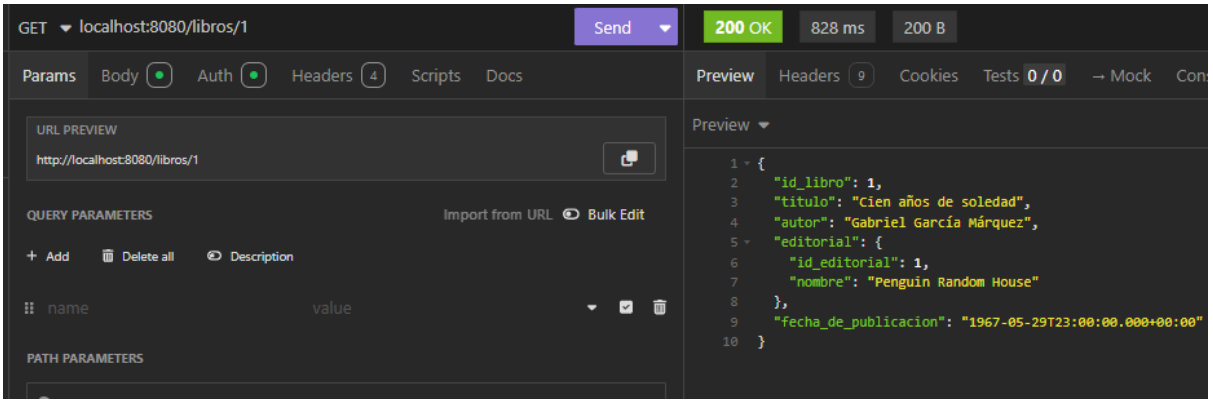
Si le falta algún dato:



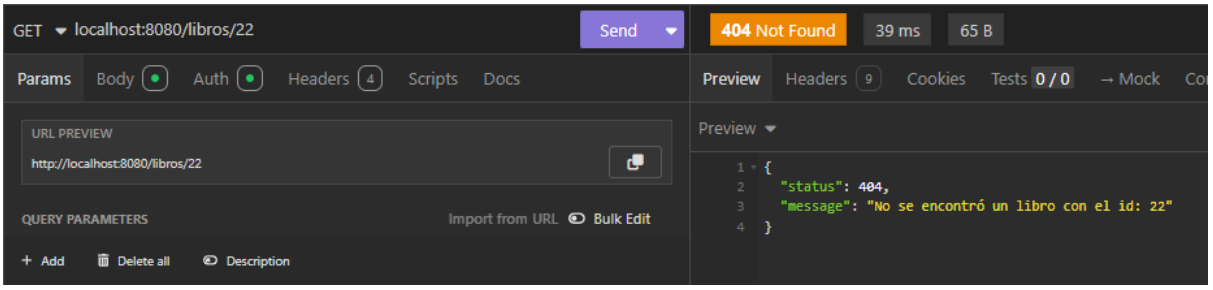
- **GET /libros**: Lista todos los libros.



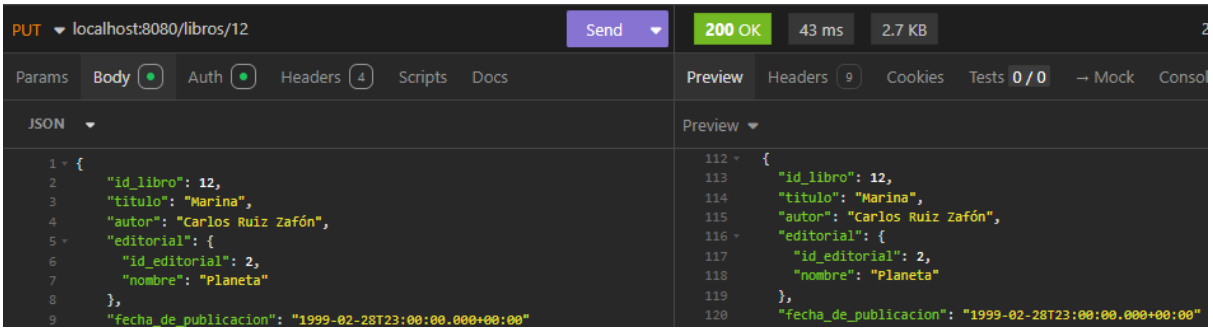
- **GET /libros/{id}**: Obtiene un libro por su ID.



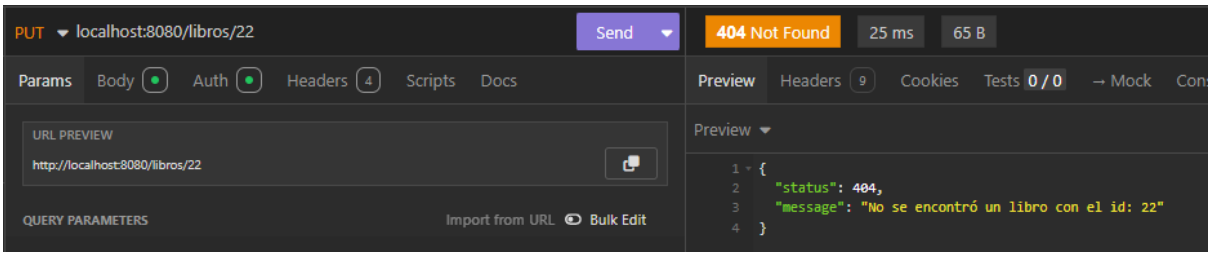
Si no existe ese libro:



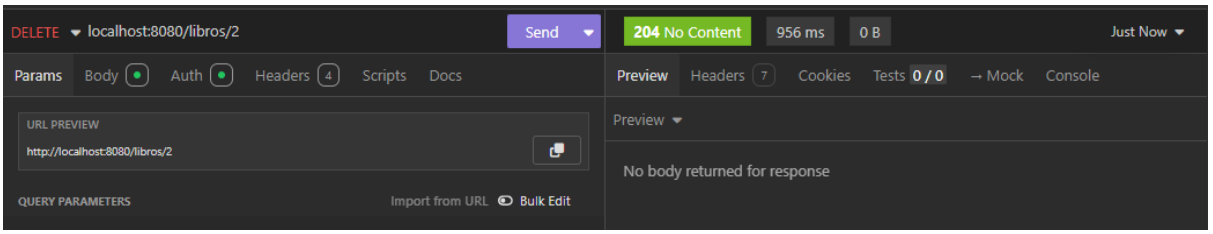
- **PUT /libros/{id}**: Actualiza un libro.



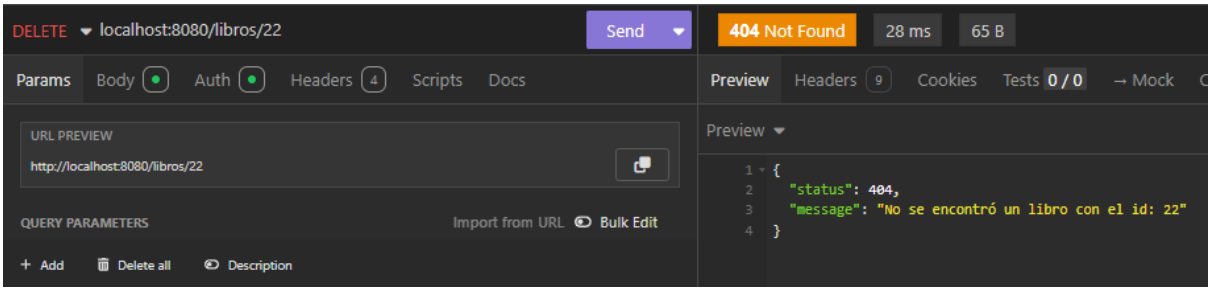
Si no existe ese libro:



- **DELETE /libros/{id}**: Elimina un libro.

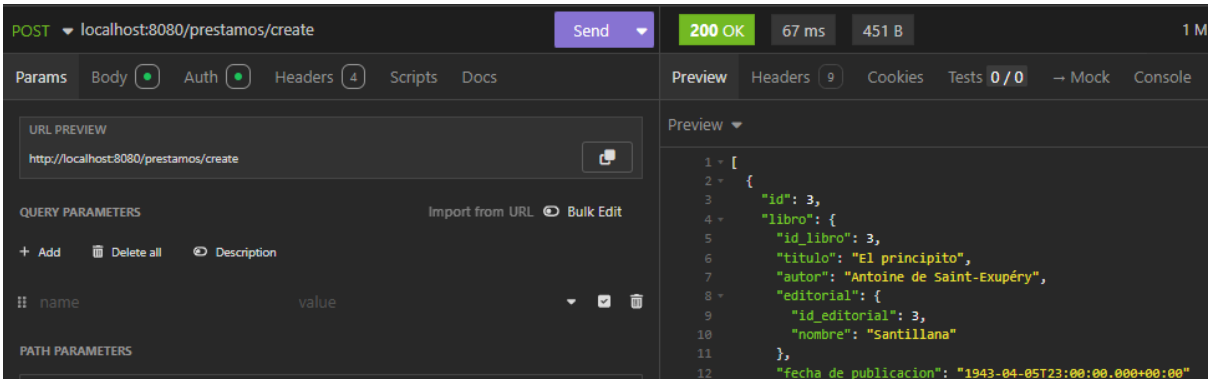


Si no existe ese libro:

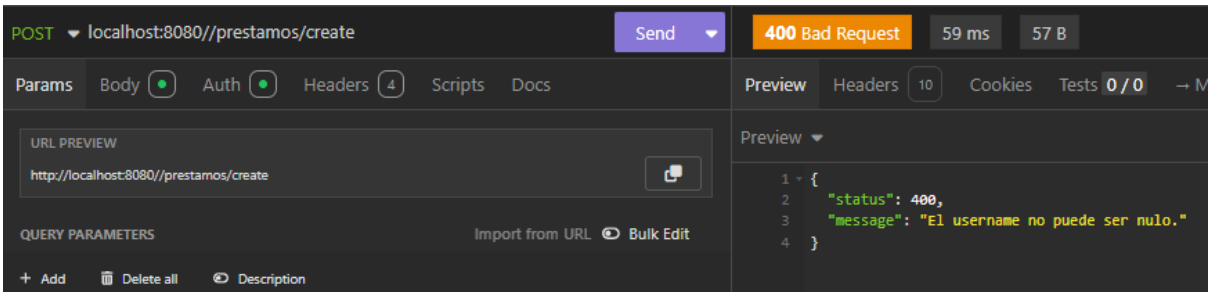


PrestamoLibroController

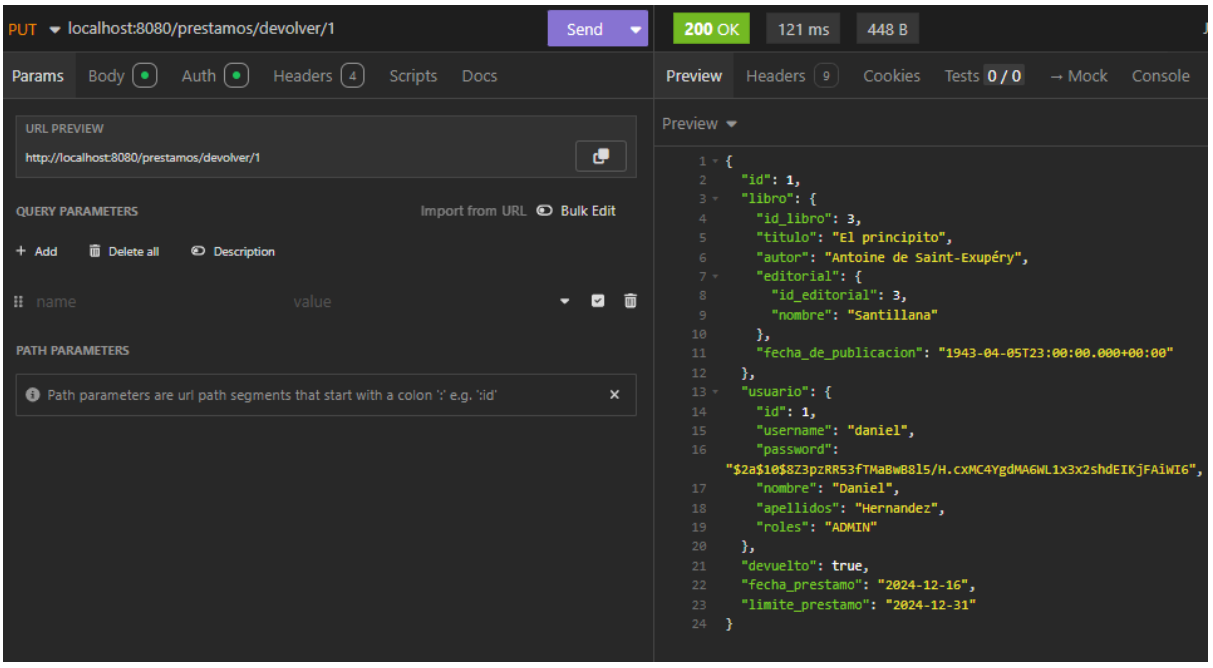
- **Endpoints**:
    - **POST /prestamos/create**: Crea un nuevo préstamo de libro.



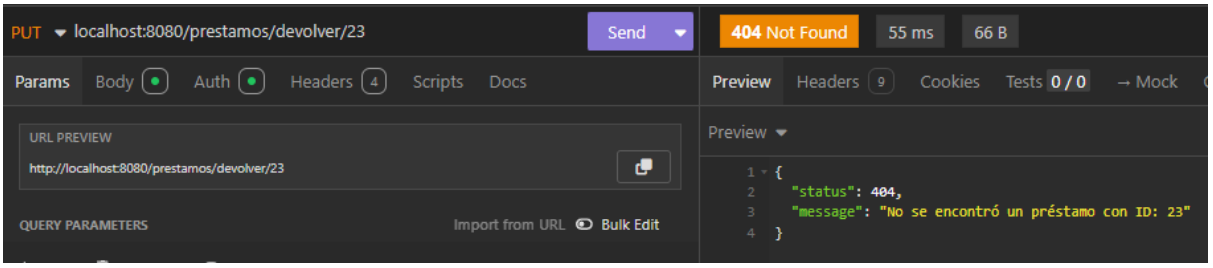
Si algún valor es incorrecto:



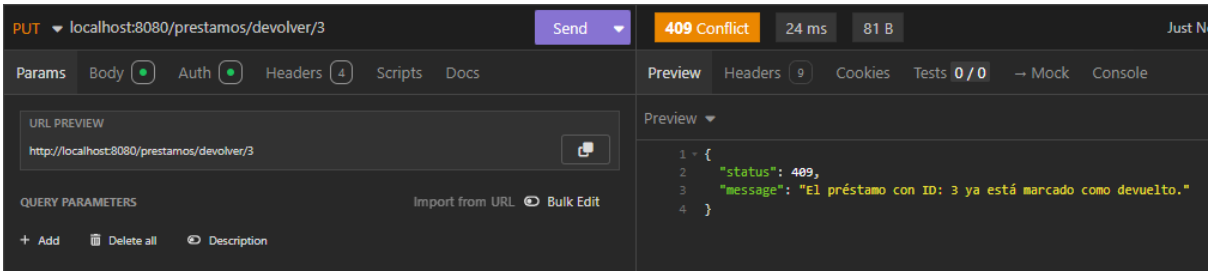
- **PUT /prestamos/devolver/{id}**: Marca un préstamo como devuelto.



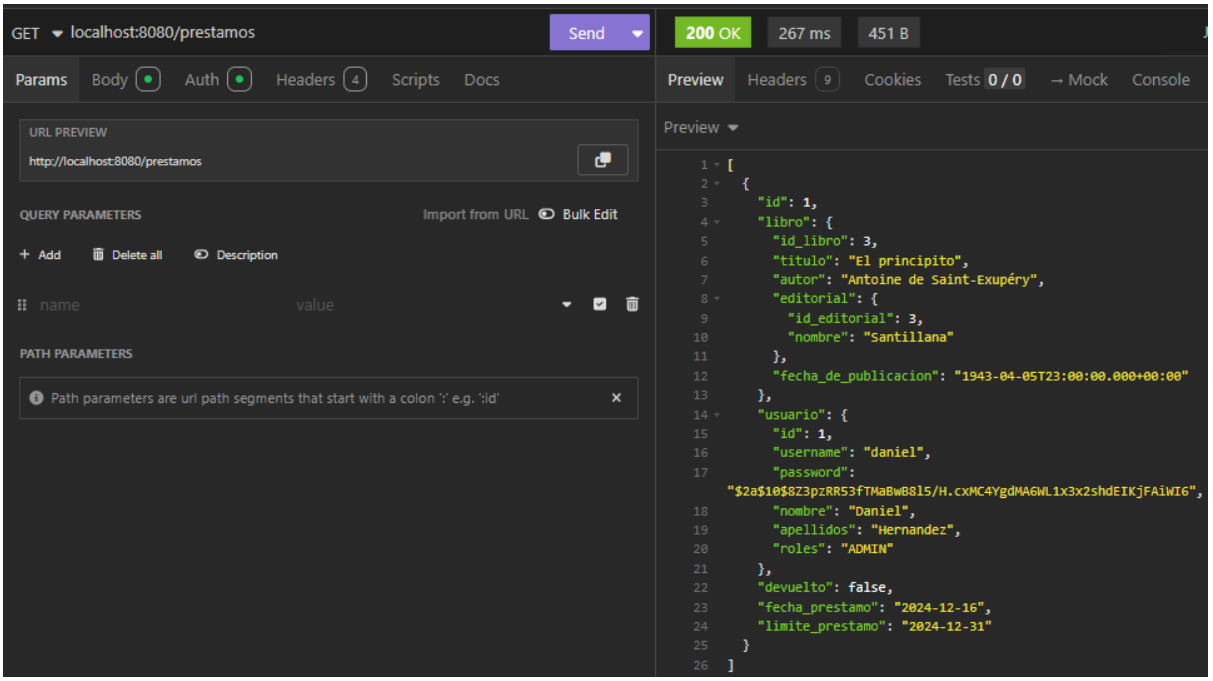
Si no existe ese préstamo:



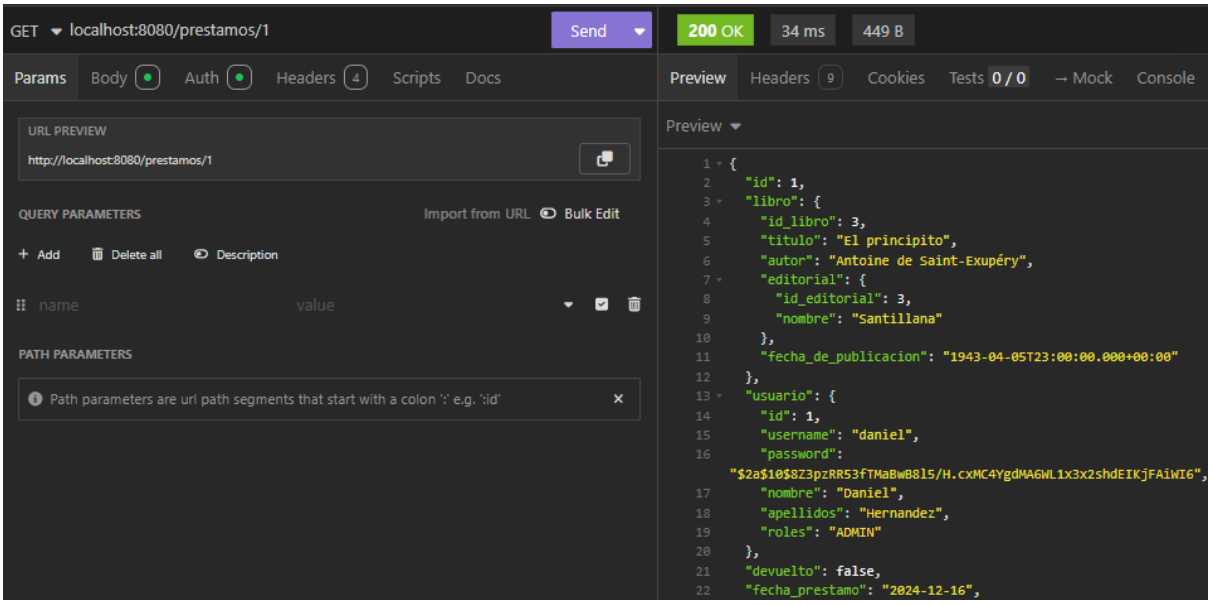
Si ya está devuelto:



- **GET /prestamos**: Lista todos los préstamos.



- **GET /prestamos/{id}**: Obtiene un préstamo por su ID.



Si no existe ese préstamo: