

# Documentación API Biblioteca

Daniel Hernández Gómez

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

## Índice:

<b>Índice:</b>	<b>1</b>
<b>1. Introducción del proyecto</b>	<b>2</b>
Título proyecto:	2
Idea del proyecto:	2
Justificación del proyecto:	2
<b>2. Descripción de las tablas</b>	<b>2</b>
Usuarios:	3
Editoriales:	3
Libros:	4
PrestamoLibro:	5
<b>3. Endpoints</b>	<b>5</b>
Usuarios:	5
Editoriales:	6
Libros:	6
Préstamos:	6
<b>4. Lógica de negocio</b>	<b>7</b>
<b>5. Excepciones y Códigos de Estado</b>	<b>7</b>
ValidationException - 400 Bad Request:	8
ConflictException - 409 Conflict:	8
NotFoundException - 404 Not Found:	8
<b>6. Restricciones de seguridad</b>	<b>8</b>
Usuarios no autenticados:	8
Usuarios autenticados:	9
<b>7. Preguntas</b>	<b>9</b>
¿Qué tecnologías has usado?	9
¿Qué es una API REST? ¿Cuáles son los principios de una API REST? ¿Dónde identificas dichos principios dentro de tu implementación?	10
¿Qué ventajas tiene realizar una separación de responsabilidades entre cliente y servidor?	11
<b>8. Pruebas</b>	<b>11</b>
UsuarioController:	11
EditorialController:	15
LibroController:	18
PrestamoLibroController:	21

# 1. Introducción del proyecto

Título proyecto:

Biblioteca Gómez

Idea del proyecto:

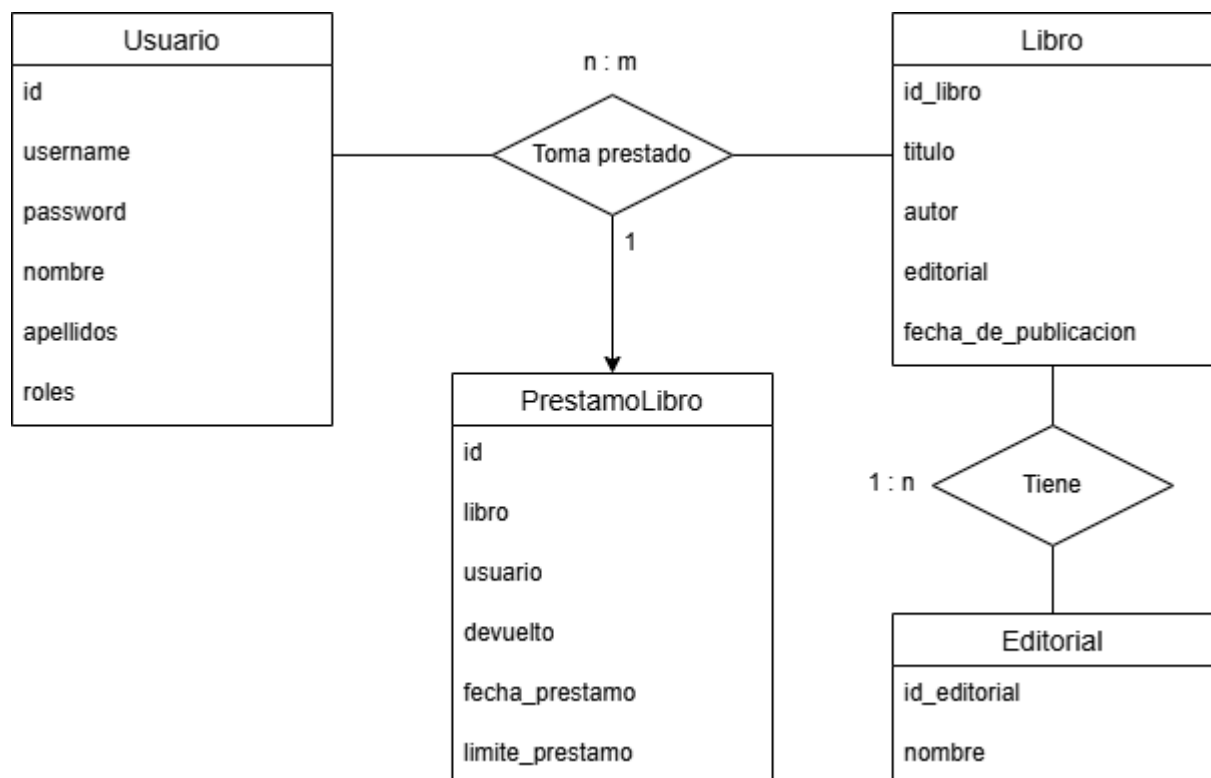
La idea del proyecto es crear una API que con una conexión a una base de datos gestione las el préstamo de libros y el registro de usuarios y libros.

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](https://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

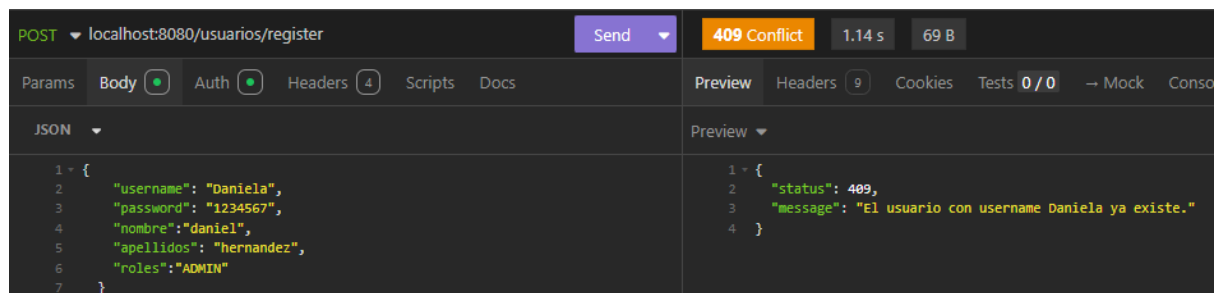
- **Endpoints:**
  - **POST /usuarios/register:** Registra un nuevo usuario.

The screenshot shows a REST client interface with the following details:

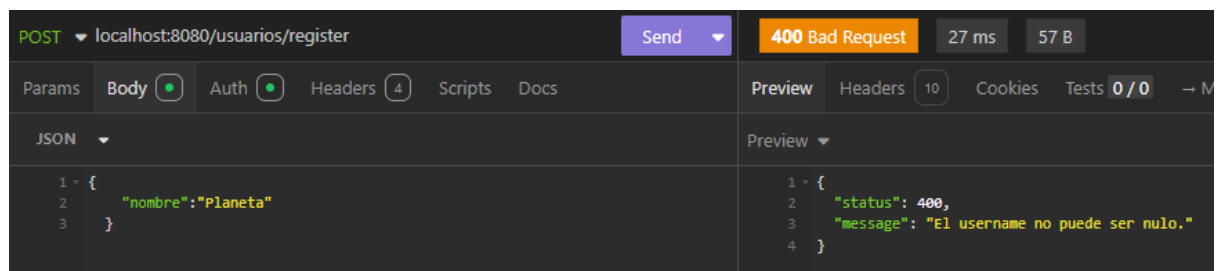
- Method:** POST
- URL:** localhost:8080/usuarios/register
- Status:** 201 Created
- Time:** 478 ms
- Size:** 162 B
- Body (JSON):**

```
1 {
2   "id": 3,
3   "username": "daniela",
4   "password": "1234",
5   "nombre": "Daniela",
6   "apellidos": "Hernandez",
7   "roles": "ADMIN"
8 }
```
- Preview:** The same JSON object is displayed in a preview format.

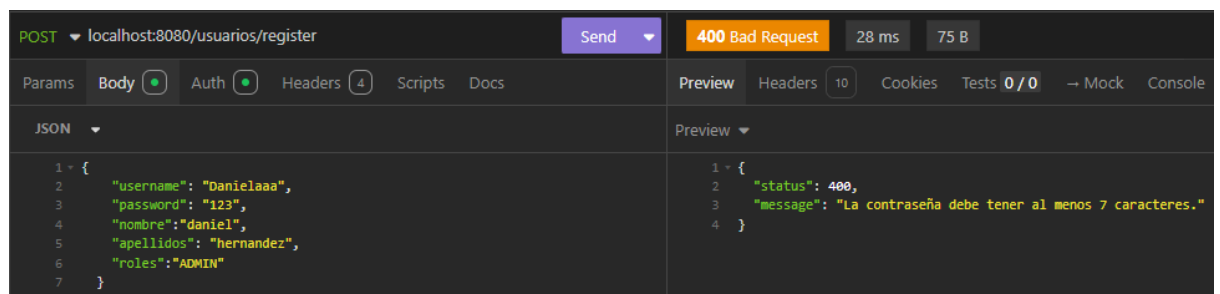
Si ya existe un usuario con ese username:



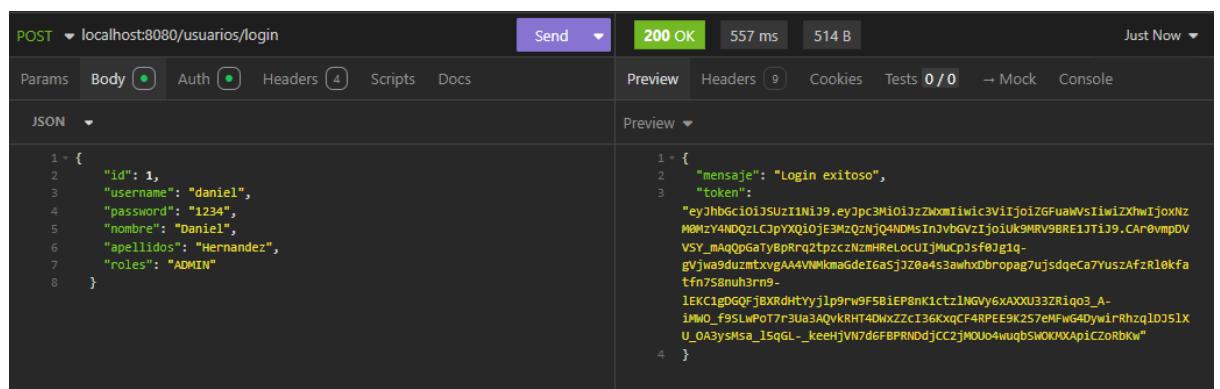
Si le faltan valores:



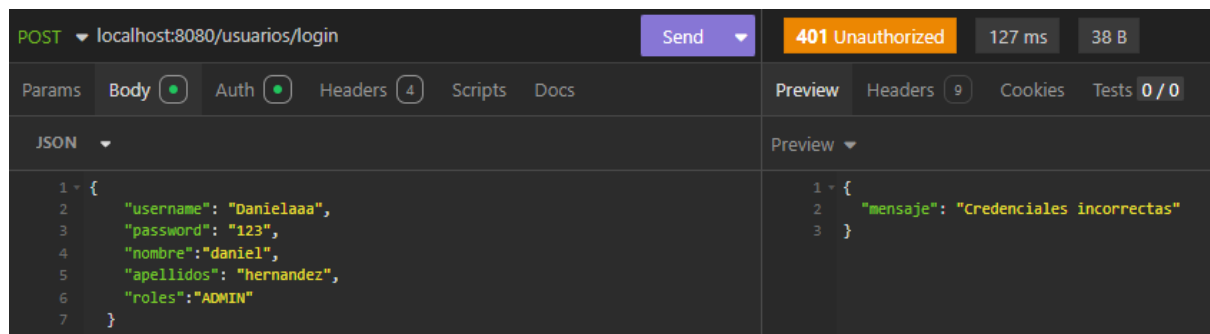
Si la contraseña no es apta:



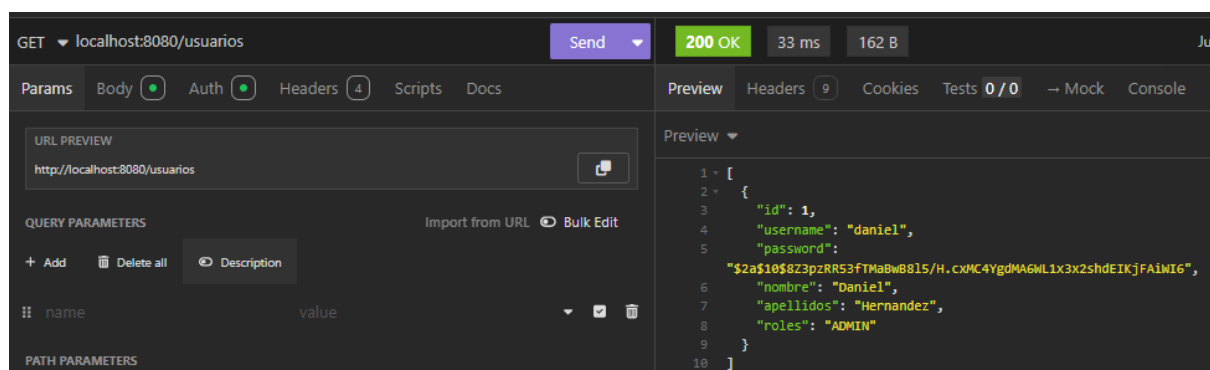
- **POST /usuarios/login:** Autentica un usuario y genera un token JWT.



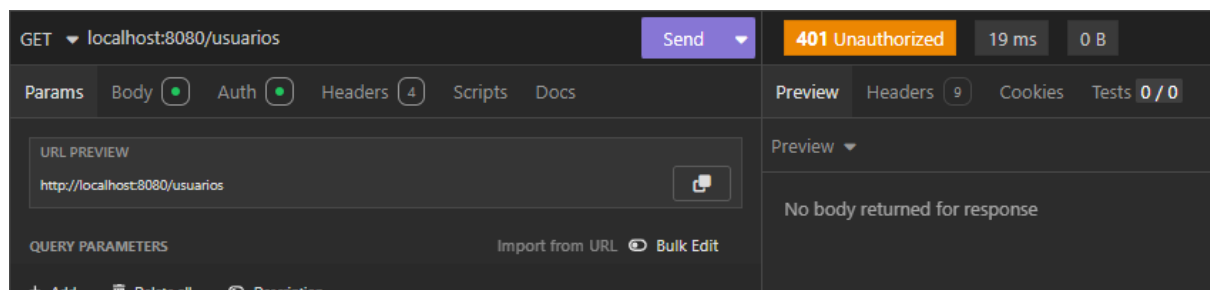
Si no existe el usuario o la contraseña es incorrecta:



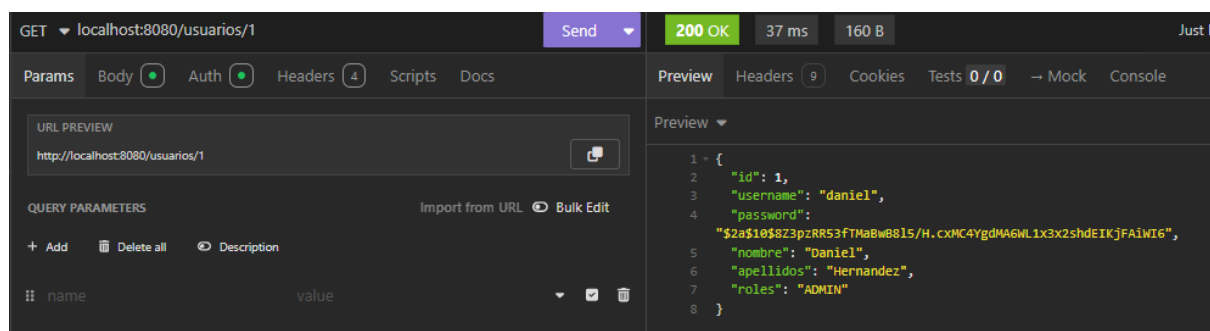
- **GET /usuarios:** Lista todos los usuarios.



si no estás autorizado:



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



Si no existe el id que buscas

GET localhost:8080/usuarios/4 Send 404 Not Found 26 ms 58 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW  
http://localhost:8080/usuarios/4

QUERY PARAMETERS  
+ Add - Delete all Description

Preview Headers 9 Cookies Tests 0/0 → Mock

```
1 {
2   "status": 404,
3   "message": "Usuario con ID 4 no encontrado."
4 }
```

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

GET localhost:8080/usuarios/4 Send 401 Unauthorized 20 ms 0 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW  
http://localhost:8080/usuarios/4

QUERY PARAMETERS  
Import from URL Bulk Edit

Preview Headers 9 Cookies Tests 0/0 → Mock

No body returned for response

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

PUT localhost:8080/usuarios/3 Send 200 OK 252 ms 162 B

Params Body Auth Headers 4 Scripts Docs

JSON

```
1 {
2   "id": 3,
3   "username": "daniela",
4   "password": "1234",
5   "nombre": "Daniela",
6   "apellidos": "Fernandez",
7   "roles": "ADMIN"
8 }
```

Preview Headers 9 Cookies Tests 0/0 → Mock Console

```
1 {
2   "id": 3,
3   "username": "daniela",
4   "password":
5     "$2a$10$0z2QV8p1sTrT58ejP588euW55eZyDZV2AIrgr1VZVrK2F7xejc0EK",
6   "nombre": "Daniela",
7   "apellidos": "Fernandez",
8   "roles": "ADMIN"
9 }
```

Si no existe ese Usuario

PUT localhost:8080/usuarios/4 Send 404 Not Found 58 ms 58 B

Params Body Auth Headers 4 Scripts Docs

JSON

```
1 {
2   "username": "Daniela",
3   "password": "1234567",
4   "nombre": "daniel",
5   "apellidos": "hernandez",
6   "roles": "ADMIN"
7 }
```

Preview Headers 9 Cookies Tests 0/0 → Mock

```
1 {
2   "status": 404,
3   "message": "Usuario con ID 4 no encontrado."
4 }
```

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

DELETE localhost:8080/usuarios/2 **204 No Content** 993 ms 0 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW  
http://localhost:8080/usuarios/2

Preview Headers 7 Cookies Tests 0/0 → Mock Console

No body returned for response

Si ese usuario no existe:

DELETE localhost:8080/usuarios/4 **404 Not Found** 478 ms 58 B

Params Body Auth Headers 4 Scripts Docs

JSON

```
1 {
2   "username": "Daniela",
3   "password": "1234567",
4   "nombre": "daniel",
5   "apellidos": "hernandez",
6   "roles": "ADMIN"
7 }
```

Preview Headers 9 Cookies Tests 0/0 → Mock Console

```
1 {
2   "status": 404,
3   "message": "Usuario con ID 4 no encontrado."
4 }
```

## EditorialController

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

POST localhost:8080/editoriales/create **201 Created** 78 ms 49 B

Params Body Auth Headers 4 Scripts Docs

JSON

```
1 {
2   "id_editorial": 3,
3   "nombre": "daniela Editoriales"
4 }
```

Preview Headers 9 Cookies Tests 0/0 → Mock Console

```
1 {
2   "id_editorial": 3,
3   "nombre": "daniela Editoriales"
4 }
```

Si ya existe una editorial con ese nombre:

POST localhost:8080/editoriales/create **409 Conflict** 26 ms 73 B

Params Body Auth Headers 4 Scripts Docs

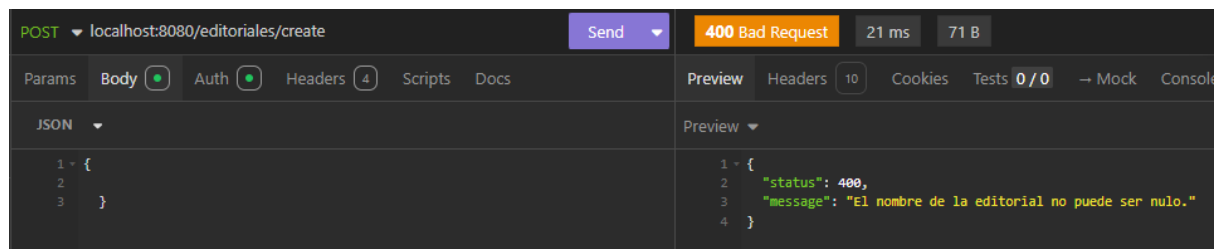
JSON

```
1 {
2   "nombre": "Planeta"
3 }
```

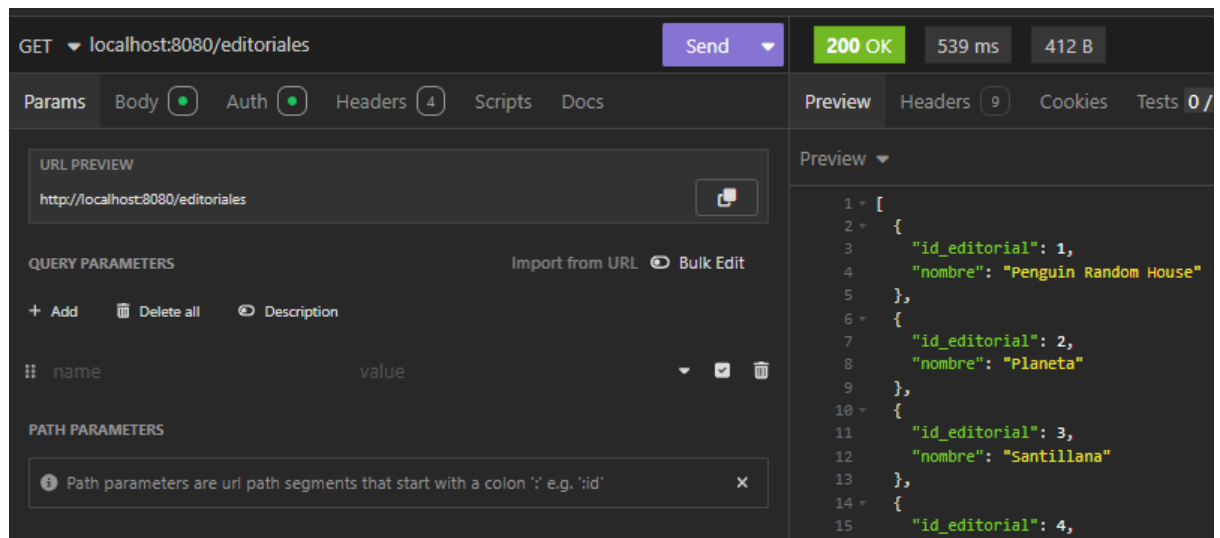
Preview Headers 9 Cookies Tests 0/0 → Mock Console

```
1 {
2   "status": 409,
3   "message": "Ya existe una editorial con el nombre: Planeta"
4 }
```

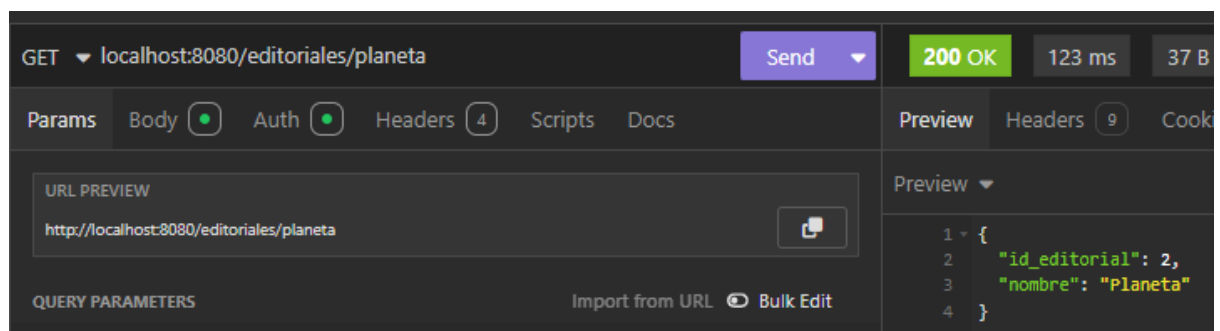
Si le faltan valores:



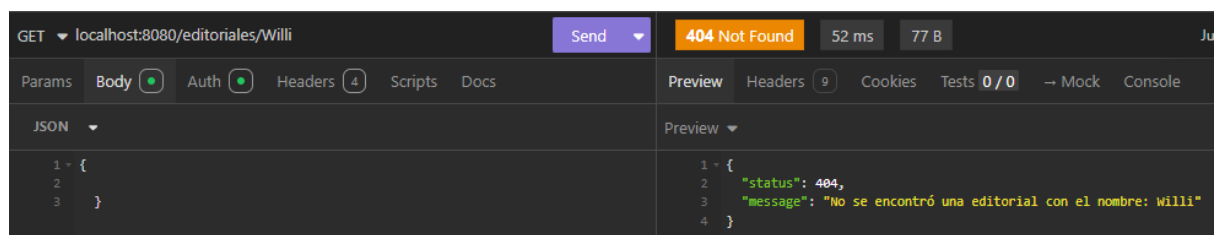
- **GET /editoriales:** Lista todas las editoriales



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



Si no existe ese nombre:





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

PUT localhost:8080/editoriales/daniela Editoriales

Send

200 OK 57 ms 58 B

Params Body Auth Headers 4 Scripts Docs

Preview Headers 9 Cookies Tests 0/0

JSON

```
1 {
2   "id_editorial": 3,
3   "nombre": "daniela Editoriales Editadas"
4 }
```

Preview

```
1 {
2   "id_editorial": 3,
3   "nombre": "daniela Editoriales Editadas"
4 }
```

Si no existe ese nombre:

PUT localhost:8080/editoriales/Willi

Send

404 Not Found 37 ms 77 B

Params Body Auth Headers 4 Scripts Docs

Preview Headers 9 Cookies Tests 0/0 Mock Console

JSON

```
1 {
2   "id_editorial": 3,
3   "nombre": "Willi"
4 }
```

Preview

```
1 {
2   "status": 404,
3   "message": "No se encontró una editorial con el nombre: Willi"
4 }
```

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

DELETE localhost:8080/editoriales/EditorialBorrar

Send

204 No Content 143 ms 0 B

Params Body Auth Headers 4 Scripts Docs

Preview Headers 7 Cookies Tests

JSON

```
1 {
2   "id_editorial": 23,
3   "nombre": "EditorialBorrar"
4 }
```

Preview

No body returned for response

Si no existe ese nombre:

DELETE localhost:8080/editoriales/Willi

Send

404 Not Found 69 ms 77 B

Params Body Auth Headers 4 Scripts Docs

Preview Headers 9 Cookies Tests 0/0 Mock Console

JSON

```
1 {
2   "id_editorial": 3,
3   "nombre": "Willi"
4 }
```

Preview

```
1 {
2   "status": 404,
3   "message": "No se encontró una editorial con el nombre: Willi"
4 }
```

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

GET localhost:8080/edoitoriales/Planeta/libros **200 OK** 43 ms 2.7 KB

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW  
http://localhost:8080/edoitoriales/Planeta/libros

QUERY PARAMETERS  
+ Add Delete all Description

Preview Headers 9 Cookies Tests 0/0 → Mock Console

```

12 {
13   "id_libro": 2,
14   "titulo": "La sombra del viento",
15   "autor": "Carlos Ruiz Zafón",
16   "editorial": {
17     "id_editorial": 2,
18     "nombre": "Planeta"
19   },
20   "fecha_de_publicacion": "2001-03-31T22:00:00.000+00:00"

```

Si no existe ese nombre:

GET localhost:8080/editoriales/Willi/libros **404 Not Found** 29 ms 77 B

Params Body Auth Headers 4 Scripts Docs

JSON

```

1 {
2   "status": 404,
3   "message": "No se encontró una editorial con el nombre: Willi"
4 }

```

Preview Headers 9 Cookies Tests 0/0 → Mock Console

```

1 {
2   "status": 404,
3   "message": "No se encontró una editorial con el nombre: Willi"
4 }

```

## LibroController

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

POST localhost:8080/libros/create **200 OK** 73 ms 2.7 KB

Params Body Auth Headers 4 Scripts Docs

JSON

```

1 {
2   "id_libro": 13,
3   "titulo": "El aleph",
4   "autor": "Jorge Luis Borges",
5   "editorial": {
6     "id_editorial": 4,
7     "nombre": "Anagrama"

```

Preview Headers 9 Cookies Tests

```

122 {
123   "id_libro": 13,
124   "titulo": "El aleph",
125   "autor": "Jorge Luis Borges",
126   "editorial": {
127     "id_editorial": 4,
128     "nombre": "Anagrama"

```

Si le falta algún dato:

POST localhost:8080/libros/create **400 Bad Request** 40 ms 55 B

Params Body Auth Headers 4 Scripts Docs

JSON

```

1 {
2   "status": 400,
3   "message": "El titulo no puede ser nulo."
4 }

```

Preview Headers 10 Cookies Tests 0/0 → N

```

1 {
2   "status": 400,
3   "message": "El titulo no puede ser nulo."
4 }

```

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

GET localhost:8080/libros

Send

200 OK 161 ms 2.7 KB

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW

http://localhost:8080/libros

QUERY PARAMETERS

Import from URL Bulk Edit

+ Add Delete all Description

name value

PATH PARAMETERS

Path parameters are url path segments that start with a colon ':' e.g. ':id'

Preview

```
1 - [  
2 - {  
3 -   "id_libro": 1,  
4 -   "titulo": "Cien años de soledad",  
5 -   "autor": "Gabriel García Márquez",  
6 -   "editorial": {  
7 -     "id_editorial": 1,  
8 -     "nombre": "Penguin Random House"  
9 -   },  
10 -  "fecha_de_publicacion": "1967-05-29T23:00:00.000+00:00"  
11 - },  
12 - {  
13 -   "id_libro": 2,  
14 -   "titulo": "La sombra del viento",  
15 -   "autor": "Carlos Ruiz Zafón",  
16 -   "editorial": {  
17 -     "id_editorial": 2,  
18 -     "nombre": "Planeta"  
19 -   },  
20 -   "fecha_de_publicacion": "2001-03-31T22:00:00.000+00:00"  
21 - },  
22 - {  
23 -   "id_libro": 3,  
24 -   "titulo": "El principito",  
25 -   "autor": "Antoine de Saint-Exupéry",  
26 -   "editorial": {  
27 -     "id_editorial": 3,  
28 -     "nombre": "Santillana"  
29 -   },  
30 -   "fecha_de_publicacion": "1943-07-01T00:00:00.000+00:00"  
31 - },  
32 - ]
```

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

GET localhost:8080/libros/1

Send

200 OK 828 ms 200 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW

http://localhost:8080/libros/1

QUERY PARAMETERS

Import from URL Bulk Edit

+ Add Delete all Description

name value

PATH PARAMETERS

Preview

```
1 - {  
2 -   "id_libro": 1,  
3 -   "titulo": "Cien años de soledad",  
4 -   "autor": "Gabriel García Márquez",  
5 -   "editorial": {  
6 -     "id_editorial": 1,  
7 -     "nombre": "Penguin Random House"  
8 -   },  
9 -   "fecha_de_publicacion": "1967-05-29T23:00:00.000+00:00"  
10 - }
```

Si no existe ese libro:

GET localhost:8080/libros/22

Send

404 Not Found 39 ms 65 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW

http://localhost:8080/libros/22

QUERY PARAMETERS

Import from URL Bulk Edit

+ Add Delete all Description

Preview

```
1 - {  
2 -   "status": 404,  
3 -   "message": "No se encontró un libro con el id: 22"  
4 - }
```

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

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8080/libros/12
- Status:** 200 OK
- Time:** 43 ms
- Size:** 2.7 KB
- Body:** JSON
- Preview:** Headers (9), Cookies, Tests (0/0), Mock, Console
- JSON Body:**

```
1 {
2   "id_libro": 12,
3   "titulo": "Marina",
4   "autor": "Carlos Ruiz Zafón",
5   "editorial": {
6     "id_editorial": 2,
7     "nombre": "Planeta"
8   },
9   "fecha_de_publicacion": "1999-02-28T23:00:00.000+00:00"
```
- Preview Body:**

```
112 {
113   "id_libro": 12,
114   "titulo": "Marina",
115   "autor": "Carlos Ruiz Zafón",
116   "editorial": {
117     "id_editorial": 2,
118     "nombre": "Planeta"
119   },
120   "fecha_de_publicacion": "1999-02-28T23:00:00.000+00:00"
```

Si no existe ese libro:

The screenshot shows a REST client interface with the following details:

- Method:** PUT
- URL:** localhost:8080/libros/22
- Status:** 404 Not Found
- Time:** 25 ms
- Size:** 65 B
- Body:** Body (selected)
- URL Preview:** http://localhost:8080/libros/22
- Preview:** Headers (9), Cookies, Tests (0/0), Mock, Console
- Preview Body:**

```
1 {
2   "status": 404,
3   "message": "No se encontró un libro con el id: 22"
4 }
```

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

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/libros/2
- Status:** 204 No Content
- Time:** 956 ms
- Size:** 0 B
- Body:** Body (selected)
- URL Preview:** http://localhost:8080/libros/2
- Preview:** Headers (7), Cookies, Tests (0/0), Mock, Console
- Preview Body:** No body returned for response

Si no existe ese libro:

The screenshot shows a REST client interface with the following details:

- Method:** DELETE
- URL:** localhost:8080/libros/22
- Status:** 404 Not Found
- Time:** 28 ms
- Size:** 65 B
- Body:** Body (selected)
- URL Preview:** http://localhost:8080/libros/22
- Preview:** Headers (9), Cookies, Tests (0/0), Mock, Console
- Preview Body:**

```
1 {
2   "status": 404,
3   "message": "No se encontró un libro con el id: 22"
4 }
```

## PrestamoLibroController

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

POST localhost:8080/prestamos/create

Send

200 OK 67 ms 451 B 1 M

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW

http://localhost:8080/prestamos/create

QUERY PARAMETERS

Import from URL Bulk Edit

+ Add Delete all Description

name value

PATH PARAMETERS

Preview Headers 9 Cookies Tests 0/0 Mock Console

```
1 - [  
2 - {  
3 -   "id": 3,  
4 -   "libro": {  
5 -     "id_libro": 3,  
6 -     "titulo": "El principito",  
7 -     "autor": "Antoine de Saint-Exupéry",  
8 -     "editorial": {  
9 -       "id_editorial": 3,  
10 -      "nombre": "Santillana"  
11 -    },  
12 -    "fecha_de_publicacion": "1943-04-05T23:00:00.000+00:00"
```

Si algún valor es incorrecto:

POST localhost:8080/prestamos/create

Send

400 Bad Request 59 ms 57 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW

http://localhost:8080/prestamos/create

QUERY PARAMETERS

Import from URL Bulk Edit

+ Add Delete all Description

Preview Headers 10 Cookies Tests 0/0 Mock Console

```
1 - {  
2 -   "status": 400,  
3 -   "message": "El username no puede ser nulo."  
4 - }
```

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

PUT localhost:8080/prestamos/devolver/1

Send

200 OK 121 ms 448 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW

http://localhost:8080/prestamos/devolver/1

QUERY PARAMETERS

Import from URL Bulk Edit

+ Add Delete all Description

name value

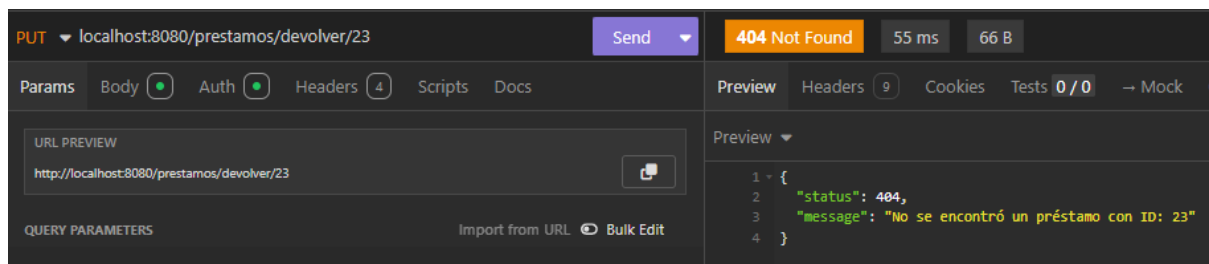
PATH PARAMETERS

Path parameters are url path segments that start with a colon ':' e.g. 'id'

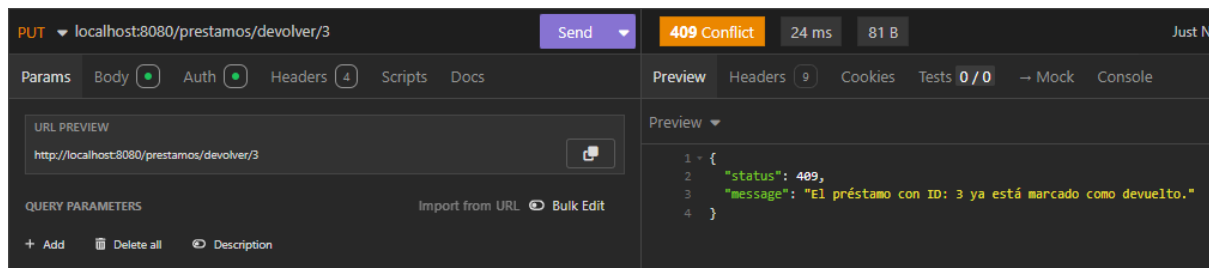
Preview Headers 9 Cookies Tests 0/0 Mock Console

```
1 - {  
2 -   "id": 1,  
3 -   "libro": {  
4 -     "id_libro": 3,  
5 -     "titulo": "El principito",  
6 -     "autor": "Antoine de Saint-Exupéry",  
7 -     "editorial": {  
8 -       "id_editorial": 3,  
9 -       "nombre": "Santillana"  
10 -    },  
11 -     "fecha_de_publicacion": "1943-04-05T23:00:00.000+00:00"  
12 -   },  
13 -   "usuario": {  
14 -     "id": 1,  
15 -     "username": "daniel",  
16 -     "password":  
17 -       "$2a$10$8Z3pZRR53fTmaBwB815/H.cXMC4YgdMA6WL1X3XshdEIKjFAiWI6",  
18 -     "nombre": "Daniel",  
19 -     "apellidos": "Hernandez",  
20 -     "roles": "ADMIN"  
21 -   },  
22 -   "devuelto": true,  
23 -   "fecha_prestamo": "2024-12-16",  
24 -   "limite_prestamo": "2024-12-31"  
25 - }
```

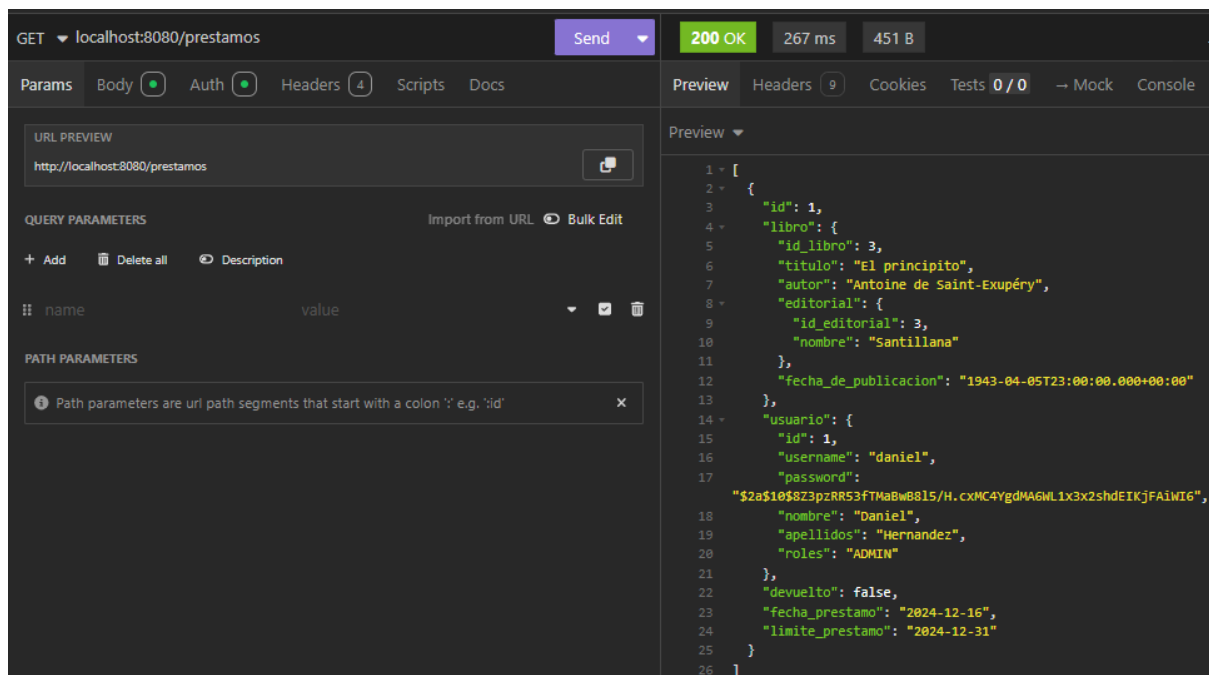
Si no existe ese préstamo:



Si ya está devuelto:



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



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

GET localhost:8080/prestamos/1 **Send** **200 OK** 34 ms 449 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW  
http://localhost:8080/prestamos/1

QUERY PARAMETERS  
+ Add Delete all Description

PATH PARAMETERS  
Path parameters are uri path segments that start with a colon ':' e.g. 'id'

Preview  
1 {  
2 "id": 1,  
3 "libro": {  
4 "id\_libro": 3,  
5 "titulo": "El principito",  
6 "autor": "Antoine de Saint-Exupéry",  
7 "editorial": {  
8 "id\_editorial": 3,  
9 "nombre": "Santillana"  
10 },  
11 "fecha\_de\_publicacion": "1943-04-05T23:00:00.000+00:00"  
12 },  
13 "usuario": {  
14 "id": 1,  
15 "username": "daniel",  
16 "password":  
17 "\$2a\$10\$8Z3pzRR53fTMaBwB815/H.cxMC4YgdMAGWL1x3x2shdEIKjFAiWi6",  
18 "nombre": "Daniel",  
19 "apellidos": "Hernandez",  
20 "roles": "ADMIN"  
21 },  
22 "devuelto": false,  
23 "fecha\_prestamo": "2024-12-16",

Si no existe ese préstamo:

GET localhost:8080/prestamos/22 **Send** **404 Not Found** 79 ms 66 B

Params Body Auth Headers 4 Scripts Docs

URL PREVIEW  
http://localhost:8080/prestamos/22

QUERY PARAMETERS  
+ Add Delete all Description

Preview  
1 {  
2 "status": 404,  
3 "message": "No se encontró un préstamo con ID: 22"  
4 }