



SISTEMAS ORIENTADOS A SERVICIOS

Diseño e implementación de un servicio
web RESTful

AUTORES

Ana García López de Asiaín
Alejandro Becerra Tapia
Alfonso Marín Mite

TABLA DE CONTENIDOS

1. INTRODUCCIÓN	III
2. ACLARACIONES Y AMBIGÜEDADES	IV
2.1 CAMBIOS DE LA SEGUNDA ENTREGA:	IV
3. RECURSOS Y OPERACIONES	VI
3.1 USUARIOS	VI
3.2 LIBROS	VIII
3.3 PRÉSTAMOS	IX
4. PERSISTENCIA DE LOS DATOS	XI
4.1 DIAGRAMA E-R	XI
4.2 EJEMPLOS DE TABLAS	XI
4.3 EJEMPLOS DE JSONS	XII
5. CAPTURAS DE LA EJECUCIÓN DE LAS OPERACIONES	XIII
5.1 USUARIOS	XIII
5.2 LIBROS	XXVII
5.3 PRESTAMOS	XXXVIII
6. CAPTURAS DE LA EJECUCIÓN DEL CLIENTE	XLVII
6.1 TEST 1 - CREAR USUARIO, LIBRO Y PRÉSTAMO (DEVOLUCIÓN)	XLVIII
6.2 TEST 2 - USUARIO BÁSICO: CREACIÓN, CONSULTA INDIVIDUAL Y LISTADO GENERAL	XLIX
6.3 TEST 3 - PRÉSTAMO BÁSICO: CREACIÓN Y CONSULTA DE PRÉSTAMOS ACTIVOS	L
6.4 TEST 4 - AMPLIAR Y DEVOLVER PRÉSTAMO	LI
6.5 TEST 5 - FILTRADO POR TÍTULO + DISPONIBILIDAD	LII
6.6 TEST 6 - CONSULTAR PRÉSTAMOS ACTIVOS E HISTORIAL	LV
6.7 TEST 7 - AMPLIAR PRÉSTAMO	LVII
6.8 TEST 8 - CONSULTAR ACTIVIDAD DE USUARIO	LIX
7. REFERENCIAS	LXI

1. INTRODUCCIÓN

RESTful (Representational State Transfer) es un estilo de arquitectura software para el desarrollo de servicios web que permite una comunicación eficiente y escalable entre sistemas distribuidos mediante el protocolo HTTP [1]. Este paradigma define un conjunto de principios que facilitan la creación de aplicaciones web sencillas, modulares y mantenibles, utilizando métodos estándar como GET, POST, PUT y DELETE para la manipulación de recursos identificados mediante URIs.

En esta práctica se ha diseñado e implementado una API RESTful que gestiona el sistema de préstamos de libros de una biblioteca. A través de esta API, los usuarios pueden registrarse, consultar y modificar su perfil, gestionar libros (crear, listar, modificar, eliminar) y realizar operaciones clave como el préstamo, la devolución y la renovación de ejemplares.

El servicio ha sido desarrollado utilizando el lenguaje Java junto con el framework Spring Boot, y los datos se almacenan de forma persistente en una base de datos PostgreSQL. Se han incorporado además filtros de búsqueda y validaciones que garantizan el correcto funcionamiento del sistema bajo diferentes escenarios.

Como parte del desarrollo, también se ha implementado un cliente en Java que permite interactuar con la API desde un entorno de consola. Este cliente permite probar todas las operaciones disponibles en la API REST, incluyendo el manejo de parámetros y verificación de los distintos códigos de respuesta HTTP.

2. ACLARACIONES Y AMBIGÜEDADES

Durante el desarrollo de la práctica se presentaron varias ambigüedades o situaciones que requirieron decisiones de implementación o validación extra que no estaban completamente especificadas en el enunciado:

- **Eliminación de recursos relacionados:** No se especifica si al eliminar un usuario o libro se deben eliminar también sus préstamos. Se decidió mantener la integridad referencial de los datos y lanzar un error 400 Bad Request si se intenta eliminar un recurso que tiene préstamos activos vinculados. Esto obliga al usuario a gestionar primero los préstamos antes de eliminar el recurso correspondiente.
- **Manejo de filtros en consultas:** En operaciones como la búsqueda de libros o préstamos activos por fecha, no se define si los filtros deben ser obligatorios o opcionales. Se implementó la lógica para que el sistema funcione correctamente con o sin parámetros, y que valide los formatos de entrada (por ejemplo, fechas en formato yyyy-MM-dd).
- **Mensajes de error personalizados:** No se establecen mensajes de error concretos para los distintos fallos. Por claridad y experiencia de usuario, se añadieron mensajes claros como "Usuario no encontrado", "No se puede eliminar un libro con préstamos activos" o "Formato de fecha inválido", siguiendo buenas prácticas de desarrollo de APIs RESTful.
- **Campos opcionales en las entidades:** Algunos atributos como `penaltyUntil` o `returnDate` pueden ser null, lo cual no se aclara expresamente en el enunciado. Se consideró válido, y el modelo de datos fue preparado para admitir esos valores nulos sin errores.
- **Formato esperado de respuestas:** En algunos endpoints (como `/usuarios/{id}/actividad`), no se especifica claramente el formato de la respuesta. Se optó por devolver un objeto compuesto que agrupa la información del usuario, la lista de préstamos activos y los últimos préstamos devueltos, con un formato coherente con las respuestas HAL (HATEOAS).

2.1 Cambios de la segunda entrega:

- Cambio de nombres en las operaciones de préstamos; `/estado` y `/fecha`, que anteriormente y de forma errónea eran verbos (`/devolver` y `/ampliar`).
- Modificación de los códigos de estado devueltos. Se han corregido los códigos HTTP que devuelve el backend en cada operación en algunos casos en los que los habíamos puesto mal como:
El préstamo con usuario penalizado, `userId` inexistente en búsqueda y la petición sin resultado, pero válida.

- Cuerpos devueltos por las operaciones POST. En la versión anterior las operaciones POST tanto de usuarios, libros y préstamos, devolvían solo el código de estado. En la versión actual, el servidor devuelve el código, un cuerpo JSON con el objeto creado y enlaces HATEOAS en los préstamos a estado y fecha.
- El Cliente al completo. Dado que era demasiado simple e incompleto, lo hemos vuelto a plantear desde el inicio haciendo varios tests, donde en cada test se ejecutan varias operaciones para comprobar el correcto funcionamiento del servicio.

3. RECURSOS Y OPERACIONES

3.1 Usuarios

DESCRIPCION	Obtener lista paginada de usuarios
METODO	GET
URI	/usuarios
CADENA DE CONSULTA	?page=0&size=10
CUERPO	-
DEVUELVE	200 OK (con array JSON) 400 Bad Request (si paginacion inválida)

DESCRIPCION	Obtiene la informacion de un usuario por su ID
METODO	GET
URI	/usuarios/{id}
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	200 OK 404 Not Found

DESCRIPCION	Crear un nuevo usuario
METODO	POST
URI	/usuarios
CADENA DE CONSULTA	-
CUERPO	json { "username": "juan.perez", "registrationNumber": "12345", "email": "juan.perez@example.com", "birthDate": "1990-01-01" }
DEVUELVE	201 Created Location: /usuarios/{id} 400 Bad Request

DESCRIPCION	Actualizar informacion de un usuario
METODO	PUT
URI	/usuarios/{id}
CADENA DE CONSULTA	-
CUERPO	json { "username": "juan.perez.actualizado", "registrationNumber": "12345", "email": "juan.perez.actualizado@example.com", "birthDate": "1990-01-01" }

	}
DEVUELVE	200 OK 204 No Content 400 Bad Request 404 Not Found

DESCRIPCION	Elimina un usuario existente
METODO	DELETE
URI	/usuarios/{id}
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	200 OK 400 Bad Request 404 Not Found

DESCRIPCION	Obtener los prestamos activos desde una fecha
METODO	GET
URI	/usuarios/{id}/prestamos
CADENA DE CONSULTA	desde=YYYY-MM-DD
CUERPO	
DEVUELVE	200 OK 404 Not Found 400 Bad Request 500 Internal Server

DESCRIPCION	Consultar el historial de prestamos devueltos
METODO	GET
URI	/usuarios/{id}/historial
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	200 OK 404 Not Found 500 Internal Server

DESCRIPCION	Consultar actividad reciente del usuario
METODO	PUT
URI	/usuarios/{id}/actividad
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	200 OK 404 Not Found 500 Internal Server

3.2 Libros

DESCRIPCION	Lista de todos los libros (paginado y filtrado)
METODO	GET
URI	/libros
CADENA DE CONSULTA	?titulo=...&disponible=true&page=0&size=10
CUERPO	-
DEVUELVE	200 OK: Lista de libros encontrados 404 Not Found: No existen libros que coincidan

DESCRIPCION	Crear un libro
METODO	POST
URI	/libros
CADENA DE CONSULTA	-
CUERPO	json{ "title": "El Quijote", "authors": "Miguel de Cervantes", "edition": "Primera", "isbn": "978-3-16-148410-0", "publisher": "Editorial Cervantes", "available": true }
DEVUELVE	201 Created: Libro creado correctamente 400 Bad Request

DESCRIPCION	Obtener libro por ID
METODO	GET
URI	/libros/{id}
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	200 OK: Libro encontrado 404 Not Found: Libro no encontrado

DESCRIPCION	Edita los datos de un libro existente
METODO	PUT
URI	/libros/{id}
CADENA DE CONSULTA	-
CUERPO	json{ "title": "El Quijote Actualizado", "authors": "Miguel de Cervantes", "edition": "Segunda",

	"isbn": "978-3-16-148410-0", "publisher": "Editorial Cervantes", "available": false }
DEVUELVE	200 OK: Libro actualizado correctamente 400 Bad Request 404 Not Found

DESCRIPCION	Eliminar un libro del sistema
METODO	DELETE
URI	/libros/{id}
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	204 No Content 400 Bad Request 404 Not Found

3.3 Préstamos

DESCRIPCION	Realiza un préstamo a un usuario de una duración de dos semanas
METODO	POST
URI	/prestamos
CADENA DE CONSULTA	-
CUERPO	<pre> json{ "userId": 1, "bookId": 10, }</pre>
DEVUELVE	201 Created 400 Bad Request 403 Forbidden 404 Not Found

DESCRIPCION	Actualizar el estado de devolución de un préstamo
METODO	PUT
URI	/prestamos/{id}/estado
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	204 No Content 400 Bad Request 404 Not Found

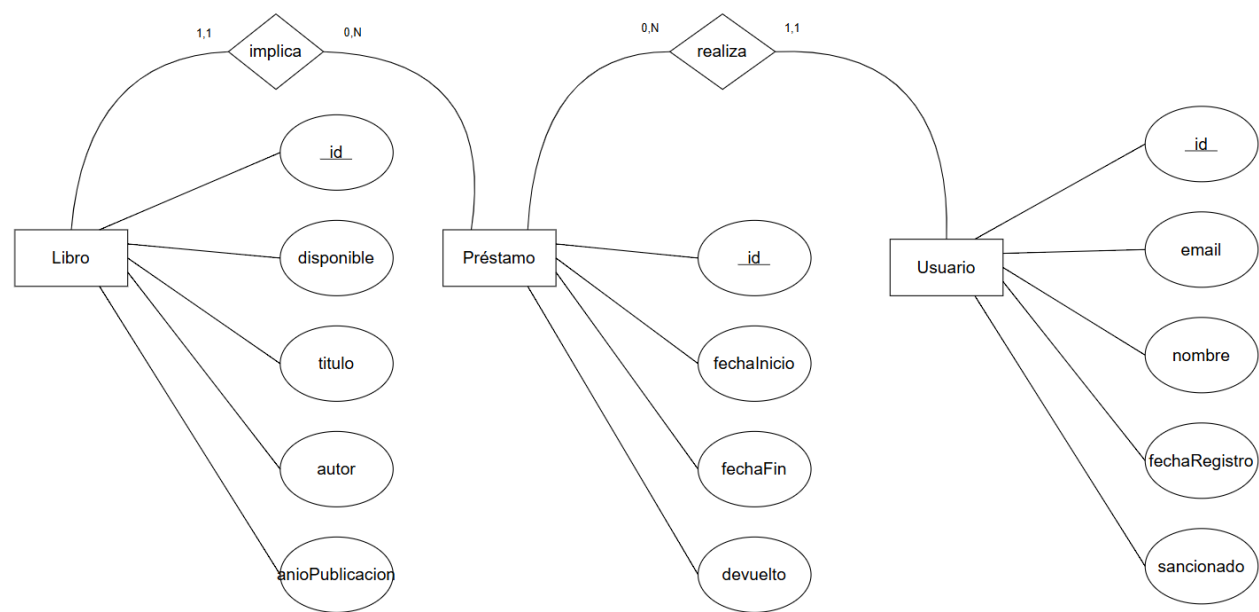
DESCRIPCION	Extender la fecha de vencimiento de un préstamo
METODO	PUT
URI	/prestamos/{id}/fecha
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	204 No Content 400 Bad Request 404 Not Found

DESCRIPCION	Obtener un préstamo por ID
METODO	GET
URI	/prestamos/{id}
CADENA DE CONSULTA	-
CUERPO	-
DEVUELVE	200 OK 404 Not Found

DESCRIPCION	Listar préstamos con filtros opcionales
METODO	GET
URI	/prestamos
CADENA DE CONSULTA	userId, current, historical, startDate, endDate
CUERPO	-
DEVUELVE	200 OK 400 Bad Request

4. PERSISTENCIA DE LOS DATOS

4.1 Diagrama E-R



4.2 EJEMPLOS DE TABLAS

USUARIOS

id	username	Registration_number	Birth_date	email
1	john_doe	REG12345	1195-03-15	John.doe@example.com
2	Jane_doe	67890	1995-05-05	jane@example.com

LIBROS

id	title	authors	edition	isbn	publisher	available
1	The Great Gatsby	F. Scott Fitzgerald	1st	9780743273565	Scribner	false
2	1984	George Orwell	1st	9780451524935	Signet Classics	true

PRESTAMOS

id	User_id	Book_id	Loan_date	Due_date	Return_date	Penalty_unit
1	1	2	2025-04-29	2025-05-13	2025-04-29	null
2	1	1	2025-04-29	2025-05-13	null	null

4.3 EJEMPLOS DE JSONS

Ejemplos de la estructuras de los JSONS usados para las pruebas realizadas

Usuarios

```
{
  "username": "john_doe",
  "registrationNumber": "REG12345",
  "birthDate": "1995-03-15",
  "email": "john.doe@example.com"
}
```

Libros

```
{
  "title": "The Great Gatsby",
  "authors": "F. Scott Fitzgerald",
  "edition": "1st",
  "isbn": "978-0743273565",
  "publisher": "Scribner",
  "available": true
}
```

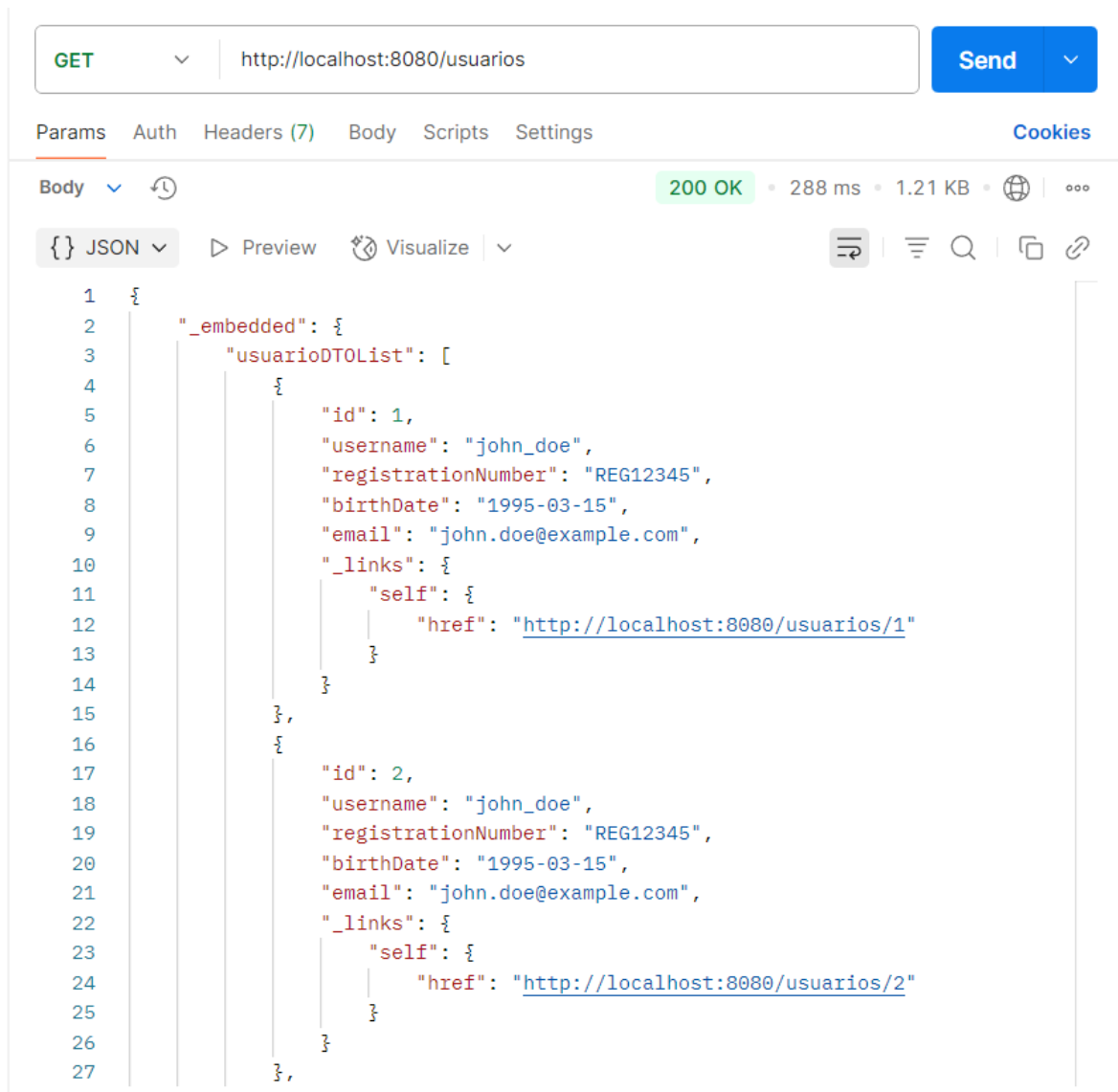
Prestamos

```
{
  "userId": 1,
  "bookId": 1
}
```

5. CAPTURAS DE LA EJECUCIÓN DE LAS OPERACIONES

5.1 Usuarios

1. GET /usuarios



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

- Method:** GET
- URL:** http://localhost:8080/usuarios
- Status:** 200 OK
- Time:** 288 ms
- Size:** 1.21 KB
- Body:** JSON

```
1 {
2   "_embedded": {
3     "usuarioDTOList": [
4       {
5         "id": 1,
6         "username": "john_doe",
7         "registrationNumber": "REG12345",
8         "birthDate": "1995-03-15",
9         "email": "john.doe@example.com",
10        "_links": {
11          "self": {
12            "href": "http://localhost:8080/usuarios/1"
13          }
14        }
15      },
16      {
17        "id": 2,
18        "username": "john_doe",
19        "registrationNumber": "REG12345",
20        "birthDate": "1995-03-15",
21        "email": "john.doe@example.com",
22        "_links": {
23          "self": {
24            "href": "http://localhost:8080/usuarios/2"
25          }
26        }
27      }
28    ]
29  }
30 }
```

<http://localhost:8080/usuarios>

El sistema responde correctamente a la solicitud mostrando todos los usuarios registrados en una lista JSON conteniendo los datos de cada uno. 200 OK indica el éxito de la operación.

GET

http://localhost:8080/usuarios?page=0&size=20

Send

ParamsAuthHeaders (7)BodyScriptsSettingsCookies

Body200 OK42 ms1.21 KB

{ } JSON


PreviewVisualize

```
51     },
52     {
53         "id": 5,
54         "username": "fallo",
55         "registrationNumber": "REG000000",
56         "birthDate": "1990-01-01",
57         "email": "john.doe@example.com",
58         "_links": {
59             "self": {
60                 "href": "http://localhost:8080/usuarios/5"
61             }
62         }
63     }
64 ]
65 },
66 "_links": {
67     "self": {
68         "href": "http://localhost:8080/usuarios?page=0&size=20"
69     }
70 },
71 "page": {
72     "size": 20,
73     "totalElements": 5,
74     "totalPages": 1,
75     "number": 0
76 }
77 }
```

<http://localhost:8080/usuarios?page=0&size=20>

Se realiza una consulta indicando la página y el tamaño y el sistema responde con los mismos usuarios de la lista junto a la información sobre los enlaces disponibles.

2. GET/usuarios/id -

 <http://localhost:8080/usuarios/1>

SaveShare

GET

<http://localhost:8080/usuarios/1>

Send

ParamsAuthHeaders (7)BodyScriptsSettingsCookies

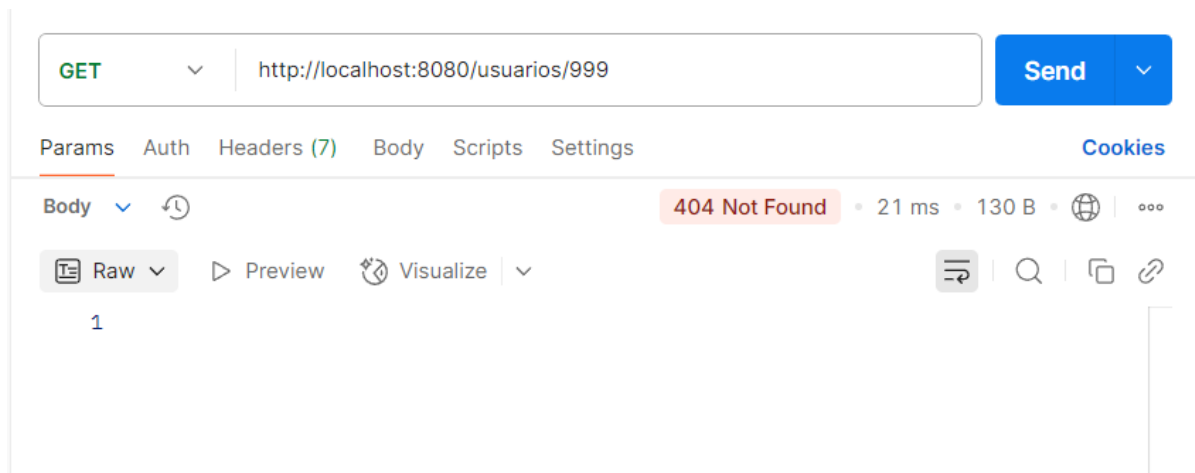
Body200 OK79 ms617 B

JSONPreviewVisualize

```
1  {
2    "id": 1,
3    "username": "john_doe",
4    "registrationNumber": "REG12345",
5    "birthDate": "1995-03-15",
6    "email": "john.doe@example.com",
7    "_links": {
8      "self": {
9        "href": "http://localhost:8080/usuarios/1"
10      },
11      "loans": {
12        "href": "http://localhost:8080/usuarios/1/prestamos{?desde}",
13        "templated": true
14      },
15      "history": {
16        "href": "http://localhost:8080/usuarios/1/historial"
17      },
18      "activity": {
19        "href": "http://localhost:8080/usuarios/1/actividad"
20      },
21      "delete": {
22        "href": "http://localhost:8080/usuarios/1"
23      }
24    }
25  }
```

<http://localhost:8080/usuarios/1>

El sistema responde correctamente a la solicitud de un usuario con un identificador, en este caso válido. Se devuelve el usuario en formato JSON confirmando con 200 OK el éxito de la operación.



<http://localhost:8080/usuarios/999>

El mensaje 404 Not Found muestra el error al intentar buscar un usuario mediante un identificador inexistente en la base de datos.

3. POST/usuarios -

POST

http://localhost:8080/usuarios

Send

ParamsAuthHeaders (9)BodyScriptsSettings

rawJSON

Beautify

```
1
2 { "username": "juan.perez", "registrationNumber": "12345",
3
4 "email": "juan.perez@example.com", "birthDate": "1990-01-01"
5
```

Body201 Created • 41 ms • 463 B

{ } JSON

PreviewVisualize

```
1 {
2   "id": 6,
3   "username": "juan.perez",
4   "registrationNumber": "12345",
5   "birthDate": "1990-01-01",
6   "email": "juan.perez@example.com",
7   "_links": {
8     "self": {
9       "href": "http://localhost:8080/usuarios/6"
10    },
11    "users": {
12      "href": "http://localhost:8080/usuarios?page=0&size=10"
13    }
14  }
15 }
```

<http://localhost:8080/usuarios>

Al realizar una solicitud de creación de usuarios con todos los campos requeridos completados, el sistema devuelve 201 Created y los datos del usuario recién creado.

POST http://localhost:8080/usuarios Send

Params Auth Headers (9) Body Scripts Settings Cookies

raw JSON Beautify

```
1 { "username": "juan.perez", "registrationNumber": "12345"
2
3
4 }
```

Body 400 Bad Request • 33 ms • 378 B •

{ } JSON Preview Visualize

```
1 {
2   "error": "Bad Request",
3   "message": "Validation failed",
4   "errors": {
5     "birthDate": "La fecha de nacimiento no puede ser nula",
6     "email": "El correo electrónico no puede estar en blanco"
7   },
8   "timestamp": "2025-06-23T15:31:21.386434200",
9   "status": 400
10 }
```

<http://localhost:8080/usuarios>

Al intentar crear un nuevo usuario con datos incompletos el sistema detecta errores de validación respondiendo con 400 BadRequest indicando en un mensaje los campos que presentan problemas y el motivo de cada fallo.

4. PUT Usuarios -

The screenshot displays a REST client interface with the following components:

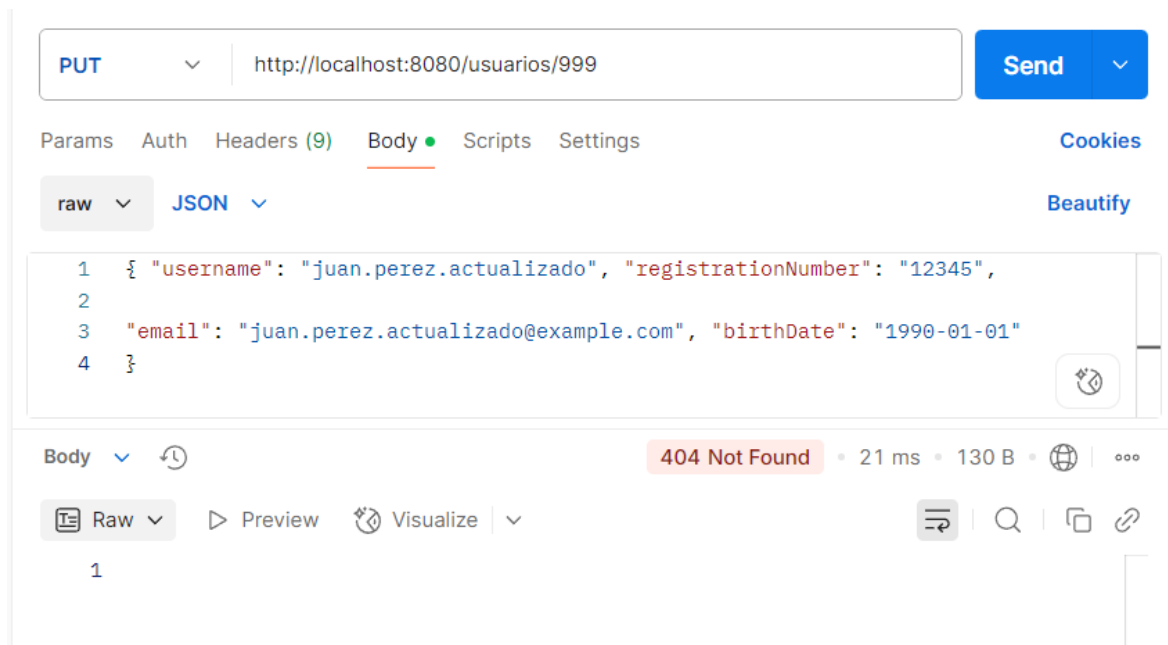
- Request Bar:** Method `PUT` and URL `http://localhost:8080/usuarios/1`. A `Send` button is on the right.
- Request Body:** The `Body` tab is selected, showing a JSON payload:

```
1 { "username": "juan.perez.actualizado", "registrationNumber": "12345",  
2  
3   "email": "juan.perez.actualizado@example.com", "birthDate": "1990-01-01"  
4 }
```
- Response Bar:** Shows `200 OK`, `62 ms`, `426 B`, and a globe icon.
- Response Body:** The `JSON` tab is selected, displaying the response payload:

```
1 {  
2   "id": 1,  
3   "username": "juan.perez.actualizado",  
4   "registrationNumber": "12345",  
5   "birthDate": "1990-01-01",  
6   "email": "juan.perez.actualizado@example.com",  
7   "_links": {  
8     "self": {  
9       "href": "http://localhost:8080/usuarios/1"  
10    },  
11    "delete": {  
12      "href": "http://localhost:8080/usuarios/1"  
13    }  
14  }  
15 }
```

<http://localhost:8080/usuarios/1>

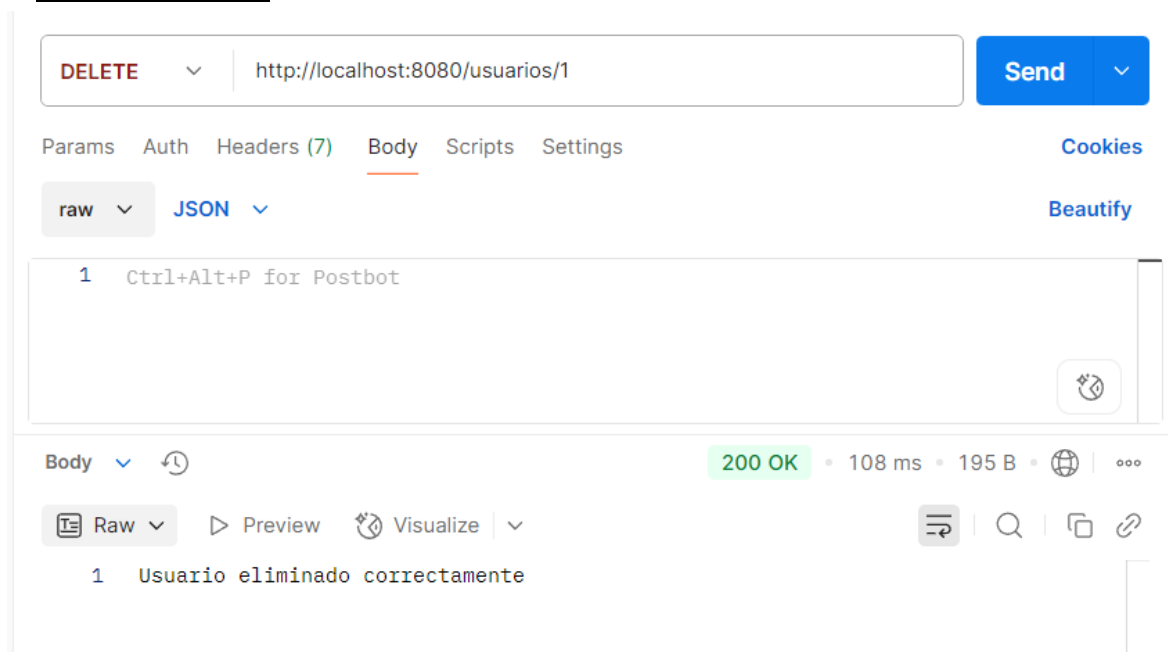
Con este comando se actualizan correctamente los datos del usuario con un identificador determinado respondiendo el sistema con 200 OK y los datos de dicho usuario actualizados en JSON.



<http://localhost:8080/usuarios/999>

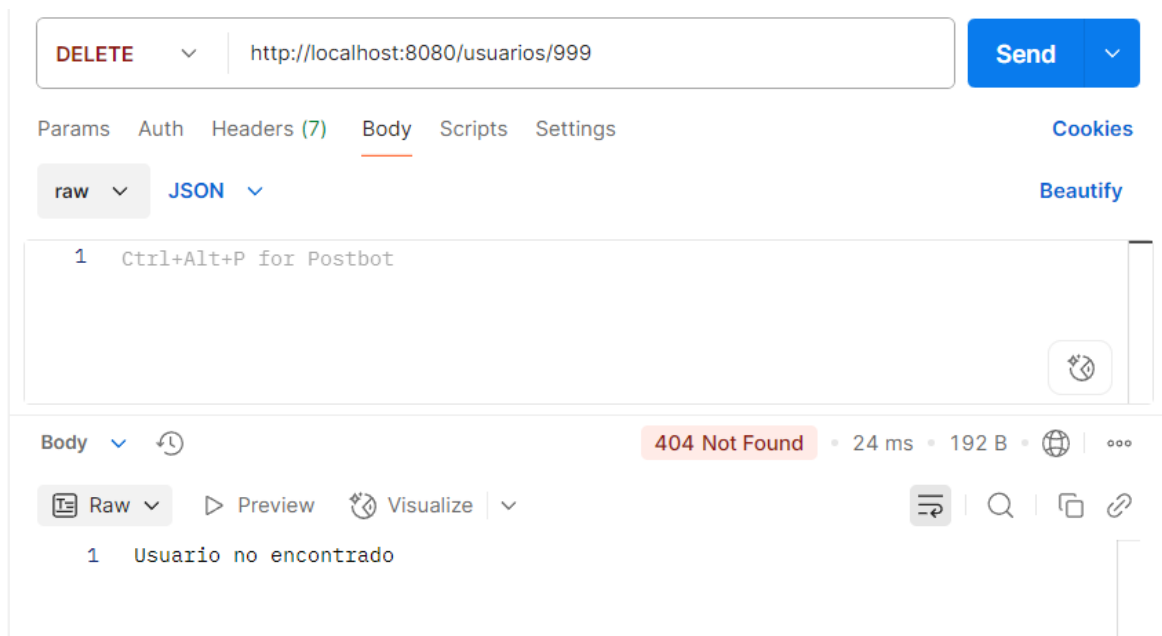
Aqui se intenta actualizar el usuario pero al ser un identificador inexistente el sistema no encuentra el recurso solicitado y devuelve 404 Not Found.

5. DELETE Usuarios -



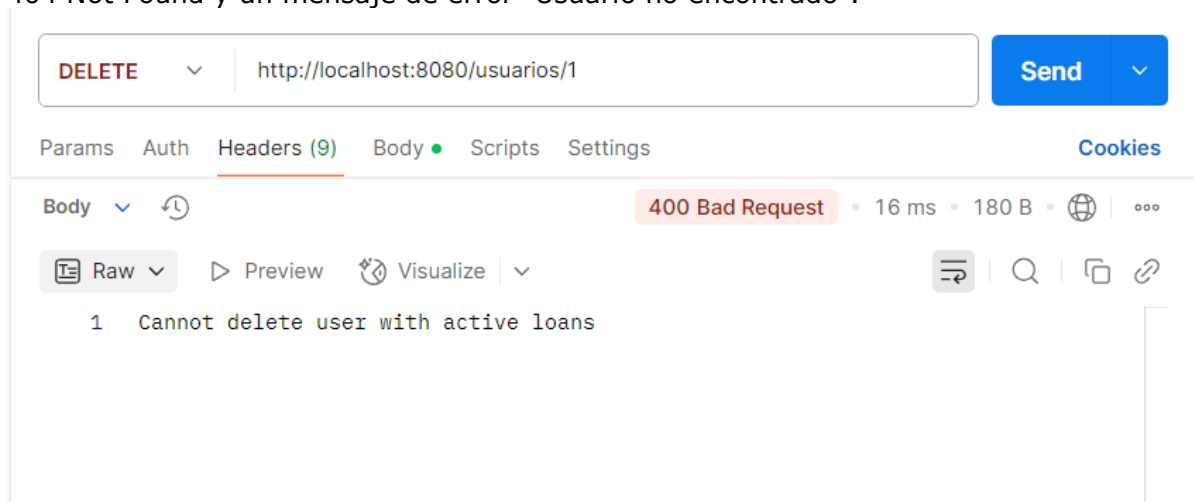
<http://localhost:8080/usuarios/1>

Se elimina correctamente un usuario con un identificador determinado y correcto. El sistema devuelve un mensaje "User deleted successfully" y el código 200 OK.



<http://localhost:8080/usuarios/999>

Al intentar eliminar a un usuario con un identificador inexistente, el sistema devuelve 404 Not Found y un mensaje de error "Usuario no encontrado".



<http://localhost:8080/usuarios/1>

En este caso se intenta eliminar a un usuario que todavía tiene préstamos activos en el sistema por lo que el servidor impide la operación y responde con el código 400 Bad Request y el mensaje de error "Cannot delete user with active loans".

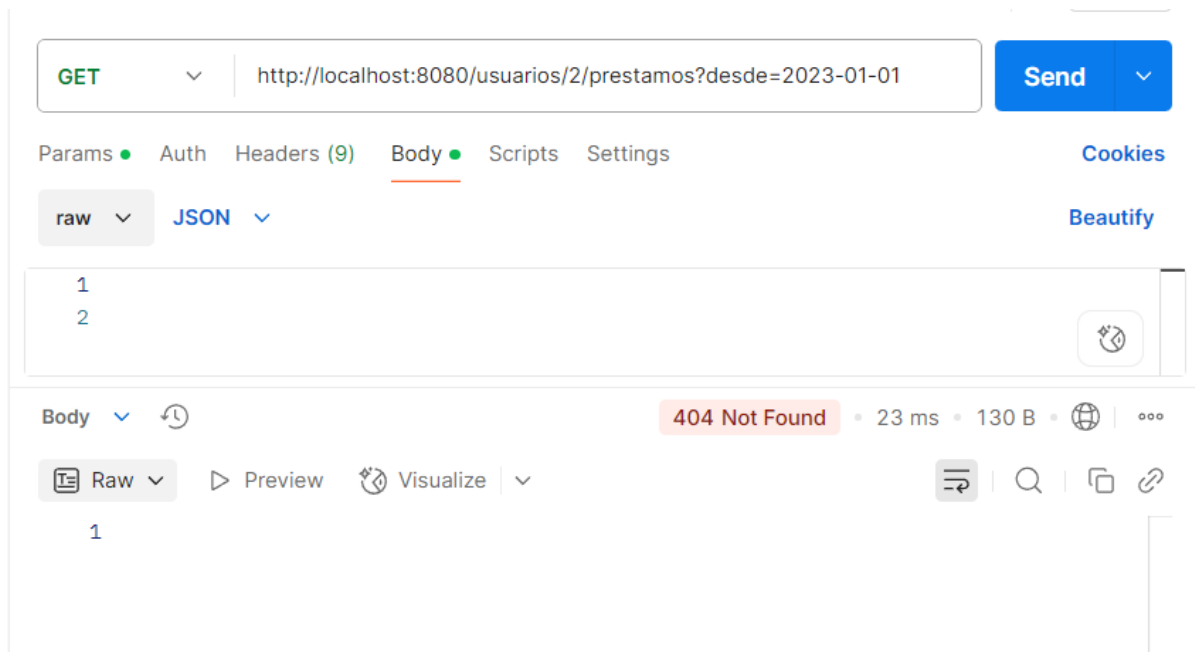
6. GET/usuarios/{id}/prestamos?desde -

The screenshot shows a REST client interface with a GET request to `http://localhost:8080/usuarios/1/prestamos?desde=2023-01-01`. The response is a 200 OK status with a 51 ms latency and 618 B of data. The response body is displayed in JSON format, showing a single active loan record for user 1 with book ID 2, loaned on 2025-06-23, and due on 2025-07-07. The JSON includes fields for loan details, return date, penalty, and HATEOAS links for returning the book and extending the loan.

```
1 {
2   "_embedded": {
3     "prestamoDTOList": [
4       {
5         "id": 2,
6         "userId": 1,
7         "bookId": 2,
8         "loanDate": "2025-06-23",
9         "dueDate": "2025-07-07",
10        "returnDate": null,
11        "penaltyUntil": null,
12        "_links": {
13          "return": {
14            "href": "http://localhost:8080/prestamos/2/estado"
15          },
16          "extend": {
17            "href": "http://localhost:8080/prestamos/2/fecha"
18          }
19        }
20      }
21    ]
22  }
```

<http://localhost:8080/usuarios/1/prestamos?desde=2023-01-01>

El sistema devuelve correctamente la lista de préstamos activos del usuario con cierto identificador desde una fecha indicada. El sistema responde con un array de datos de los préstamos y enlaces HATEOAS para devolver o ampliar cada uno de ellos.



<http://localhost:8080/usuarios/4/prestamos?desde=2023-01-01>

Al intentar obtener los prestamos de un usuario con identificador inexistente o sin prestamos activos registrados, el sistema devuelve 404 Not Found.

7. GET /usuarios/{id}/historial -

GET http://localhost:8080/usuarios/1/historial Send

Params Auth Headers (9) Body Scripts Settings Cookies

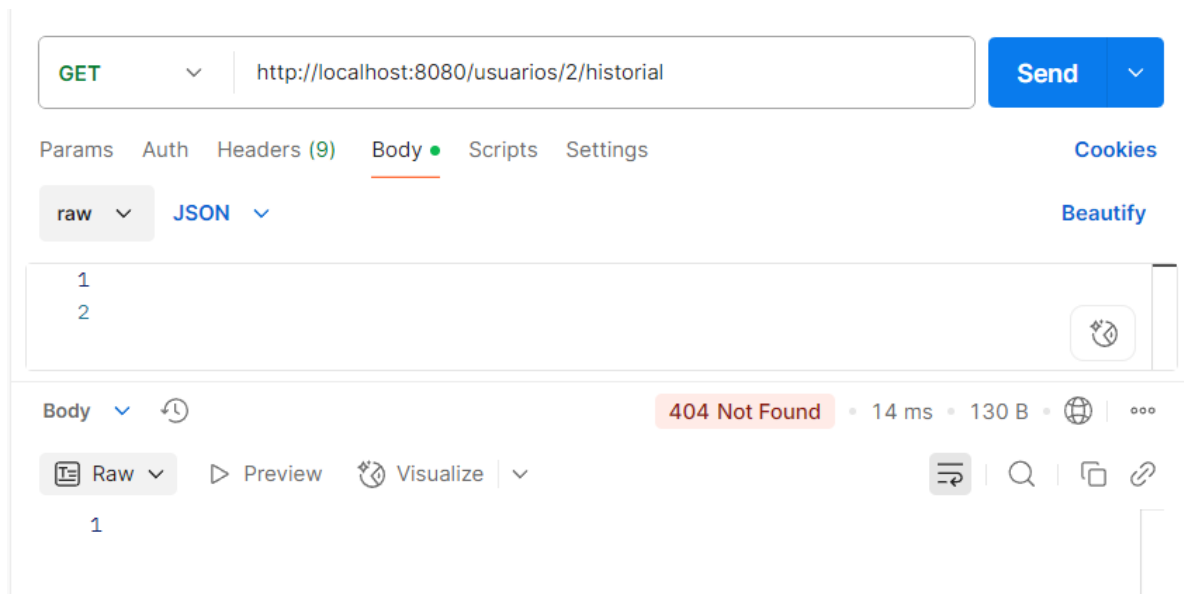
raw JSON Beautify

Body 200 OK • 47 ms • 450 B

```
{
  "_embedded": {
    "prestamoDTOList": [
      {
        "id": 2,
        "userId": 1,
        "bookId": 2,
        "loanDate": "2025-06-23",
        "dueDate": "2025-07-07",
        "returnDate": "2025-06-23",
        "penaltyUntil": null
      }
    ]
  },
  "_links": {
    "self": {
      "href": "http://localhost:8080/usuarios/1/historial"
    },
    "user": {
      "href": "http://localhost:8080/usuarios/1"
    }
  }
}
```

<http://localhost:8080/usuarios/1/historial>

El sistema devuelve correctamente el historial del usuario especificado mediante su identificador incluyendo una lista JSON con los préstamos, sus fechas de inicio, vencimiento y devolución y los enlaces relacionados.



<http://localhost:8080/usuarios/2/historial>

El sistema devuelve el error 404 Not Found al no encontrar un usuario con el identificador especificado.

8. GET /usuarios/{id}/actividad -

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

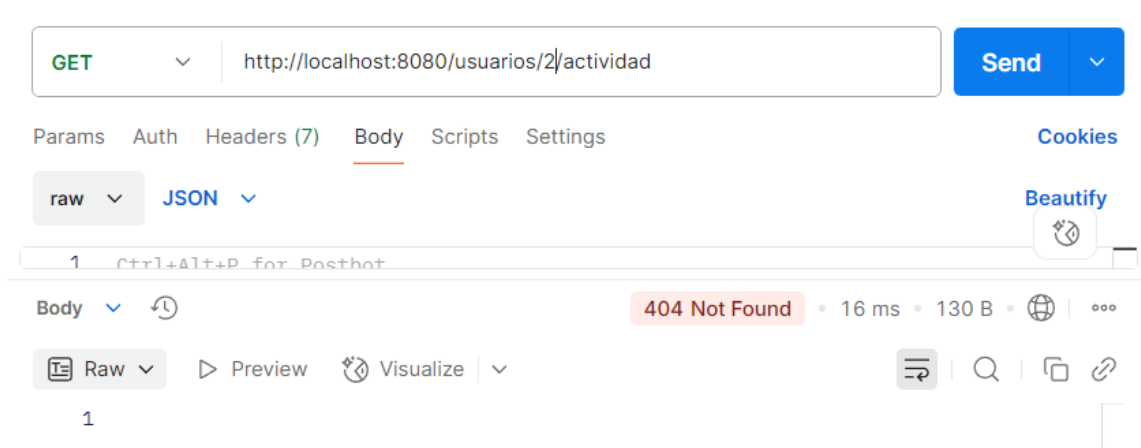
- Method:** GET
- URL:** http://localhost:8080/usuarios/1/actividad
- Status:** 200 OK
- Time:** 45 ms
- Size:** 775 B

The response body is a JSON object:

```
{
  "usuario": {
    "id": 1,
    "username": "ana_test",
    "registrationNumber": "REG99999",
    "birthDate": "1999-01-01",
    "email": "ana@example.com"
  },
  "activeLoans": [
    {
      "id": 3,
      "userId": 1,
      "bookId": 3,
      "loanDate": "2025-06-23",
      "dueDate": "2025-07-07",
      "returnDate": null,
      "penaltyUntil": null
    }
  ],
  "recentLoans": [
    {
      "id": 2,
      "userId": 1,
      "bookId": 2,
      "loanDate": "2025-06-23",
      "dueDate": "2025-07-07",
      "returnDate": "2025-06-23",
      "penaltyUntil": null
    }
  ]
}
```

<http://localhost:8080/usuarios/1/actividad>

Muestra la actividad reciente del usuario de identificador especificado devolviendo correctamente los datos del usuario, la lista de préstamos activos y los últimos préstamos devueltos.

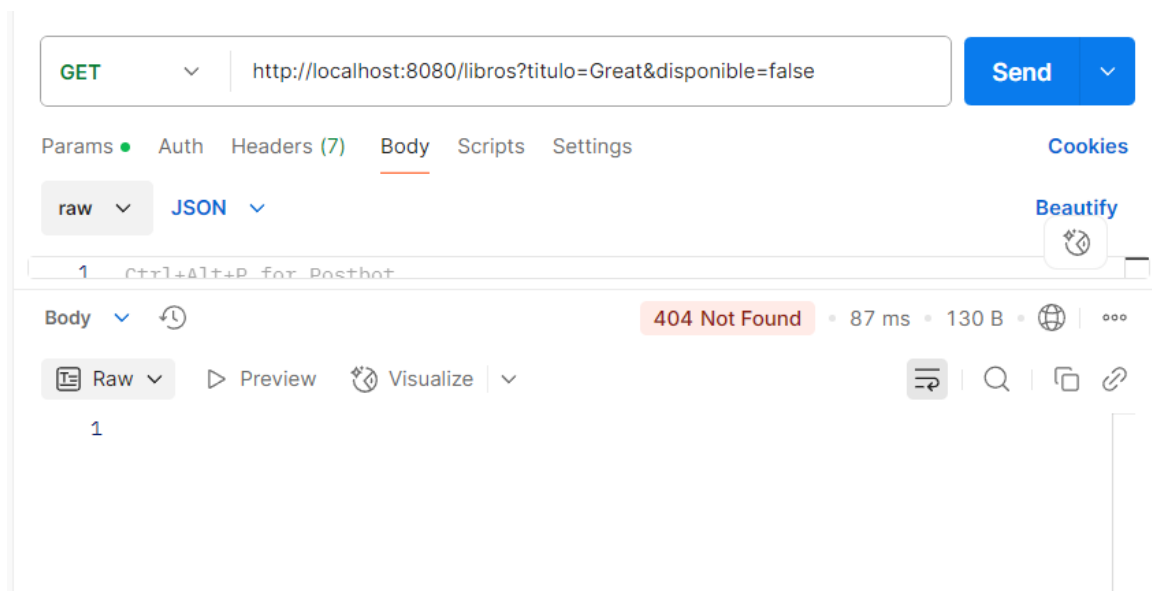


<http://localhost:8080/usuarios/2/actividad>

Como ya hemos visto anteriormente, el sistema devolverá 404 Not Found al no encontrar en la base de datos un usuario con dicho identificador.

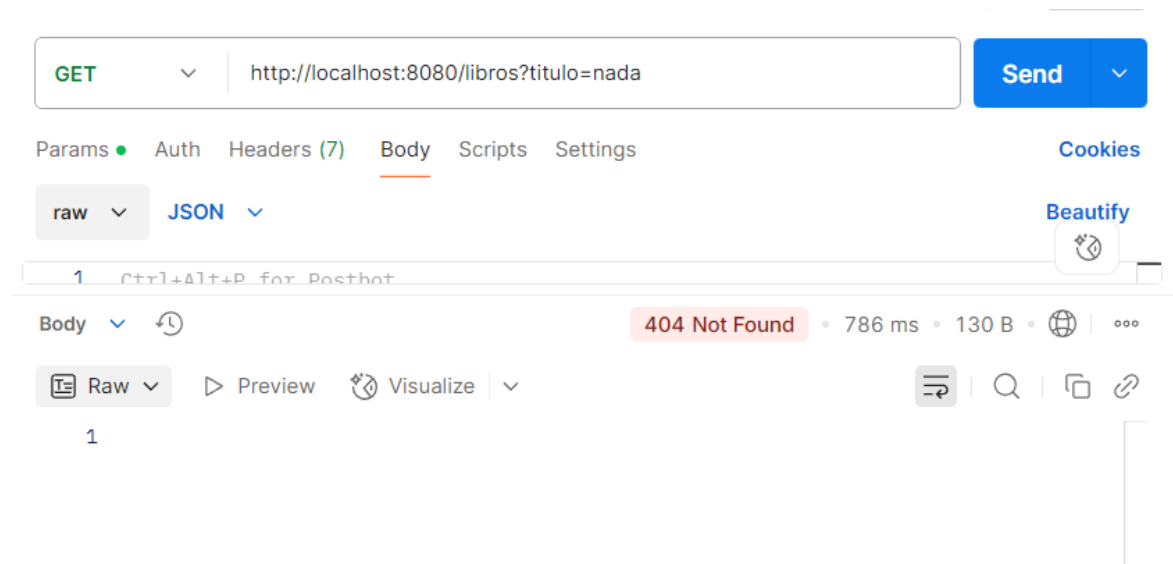
5.2 Libros

1. GET /libros?titulo=...&disponible=...&page=0&size=10



<http://localhost:8080/libros?titulo=Great&disponible=false>

Devuelve 404 Not Found porque no existe ningún libro con título Great y que no esté disponible, por ejemplo, en este caso de búsqueda. No hay coincidencias en la búsqueda combinada por título y disponibilidad.



<http://localhost:8080/libros?titulo=nada>

En este caso no existe un libro con el título de la búsqueda.

GET http://localhost:8080/libros?titulo=Great&disponible=true Send

Params Auth Headers (7) Body Scripts Settings Cookies

raw JSON Beautify

1 Ctrl+Alt+P for Postbot

Body 200 OK • 22 ms • 513 B

{ } JSON Preview Visualize

```
1 {
2   "_embedded": {
3     "libroDTOList": [
4       {
5         "id": 1,
6         "title": "The Great Gatsby",
7         "authors": "F. Scott Fitzgerald",
8         "edition": "1st",
9         "isbn": "978-0743273565",
10        "publisher": "Scribner",
11        "available": true
12      }
13    ]
14  },
15  "_links": {
16    "self": {
17      "href": "http://localhost:8080/libros?titulo=Great&disponible=true&page=0&size=20"
18    }
19  },
20  "page": {
21    "size": 20,
22    "totalElements": 1
23  }
24 }
```

<http://localhost:8080/libros?titulo=Great&disponible=true>

En este caso, según los filtros de búsqueda por título y por disponibilidad, el sistema devuelve 200 OK al sí encontrar el libro disponible.

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

- Method:** GET
- URL:** `http://localhost:8080/libros?titulo=sombra`
- Buttons:** Send, Cookies, Beautify
- Tabs:** Params, Auth, Headers (7), Body (selected), Scripts, Settings
- Body Format:** raw, JSON (selected)
- Shortcut:** Ctrl+Alt+P for Postbot
- Status:** 200 OK, 24 ms, 500 B
- JSON Response:**

```
1  {
2    "_embedded": {
3      "libroDTOList": [
4        {
5          "id": 3,
6          "title": "La sombra del viento",
7          "authors": "Carlos Ruiz Zafón",
8          "edition": "1st",
9          "isbn": "9788408053643",
10         "publisher": "Planeta",
11         "available": false
12       }
13     ],
14   },
15   "_links": {
16     "self": {
17       "href": "http://localhost:8080/libros?titulo=sombra&page=0&size=20"
18     }
19   },
20   "page": {
21     "size": 20,
22     "totalElements": 1
23   }
24 }
```

<http://localhost:8080/libros?titulo=sombra>

El sistema devuelve el libro de título "Sistemas Distribuidos" en este caso porque coincide parcialmente con el filtro de título buscado. Devuelve una lista siempre de todos los resultados coincidentes.

GET http://localhost:8080/libros?disponible=true Send

Params Auth Headers (9) Body Scripts Settings Cookies

Body 200 OK • 25 ms • 672 B

{ } JSON Preview Visualize

```
1 {
2   "_embedded": {
3     "libroDTOList": [
4       {
5         "id": 1,
6         "title": "The Great Gatsby",
7         "authors": "F. Scott Fitzgerald",
8         "edition": "1st",
9         "isbn": "978-0743273565",
10        "publisher": "Scribner",
11        "available": true
12      },
13      {
14        "id": 2,
15        "title": "El Quijote Actualizado",
16        "authors": "Miguel de Cervantes",
17        "edition": "Segunda",
18        "isbn": "978-3-16-148410-0",
19        "publisher": "Editorial Cervantes",
20        "available": true
21      }
22    ]
23  },
24  "_links": {
25    "self": {
26      "href": "http://localhost:8080/libros?disponible=true&page=0&
```

<http://localhost:8080/libros?disponible=true>

El sistema devuelve una lista completa de libros con sus datos y enlaces HATEOAS filtrada por su disponibilidad.

The screenshot shows a web browser's developer tools interface. At the top, the method is set to 'GET' and the URL is 'http://localhost:8080/libros?disponible=maybe'. A blue 'Send' button is visible. Below the URL bar, tabs for 'Params', 'Auth', 'Headers (9)', 'Body', 'Scripts', and 'Settings' are shown, with 'Body' selected. On the right, a 'Cookies' tab is also visible. The 'Body' tab displays a JSON response with a red status bar indicating '400 Bad Request'. The JSON object contains the following fields: 'timestamp' (2025-06-24), 'status' (400), 'error' (Bad Request), and 'path' (/libros). The response size is 222 B and the time taken is 47 ms. The JSON is displayed in a code editor with line numbers 1 through 6.

```
1 {  
2   "timestamp": "2025-06-24",  
3   "status": 400,  
4   "error": "Bad Request",  
5   "path": "/libros"  
6 }
```

<http://localhost:8080/libros?disponible=maybe>

El sistema devuelve 400 Bad Request dado que se ha utilizado en la petición un parámetro no válido, como es en este caso maybe.

GET

http://localhost:8080/libros?disponible=false

Send

ParamsAuthHeaders (9)BodyScriptsSettingsCookies

Body200 OK29 ms676 B

JSON

PreviewVisualize

```
1  {
2    "_embedded": {
3      "libroDTOList": [
4        {
5          "id": 3,
6          "title": "La sombra del viento",
7          "authors": "Carlos Ruiz Zafón",
8          "edition": "1st",
9          "isbn": "9788408053643",
10         "publisher": "Planeta",
11         "available": false
12       },
13       {
14         "id": 4,
15         "title": "El Quijote Actualizado",
16         "authors": "Miguel de Cervantes",
17         "edition": "Segunda",
18         "isbn": "978-3-16-148410-0",
19         "publisher": "Editorial Cervantes",
20         "available": false
21       }
22     ]
23   },
24   "_links": {
25     "self": {
26       "href": "http://localhost:8080/libros?disponible=false&page=0&size=20"
```

<http://localhost:8080/libros?disponible=false>

Filtra los libros que no están disponibles devolviéndolos correctamente en una lista junto a sus datos y enlaces útiles.

GET ▼ http://localhost:8080/libros/999 Send ▼

Params Auth Headers (7) Body Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1

Body ▼ 🕒 404 Not Found • 36 ms • 130 B • 🌐 | ⋮

📄 Raw ▼ ▶ Preview 🔗 Visualize ▼ ☰ | 🔍 | 📋 | 🔗

1

<http://localhost:8080/libros/999>

Devuelve 404 Not Found porque la petición es incorrecta y no encuentra el recurso solicitado.

2. POST/Libros -

POST ▼ http://localhost:8080/libros Send ▼

Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```

1 {
2   "title": "Cuento ejemplo",
3   "authors": "Juan Pérez",
4   "edition": "1st",
5   "isbn": "1234567890123",
6   "publisher": "Editorial ADF",

```

Body ▼ 🕒 201 Created • 138 ms • 416 B • 🌐 | ⋮

{} JSON ▼ ▶ Preview 🔗 Visualize ▼ ☰ | 🔍 | 📋 | 🔗

```

1 {
2   "id": 5,
3   "title": "Cuento ejemplo",
4   "authors": "Juan Pérez",
5   "edition": "1st",
6   "isbn": "1234567890123",
7   "publisher": "Editorial ADF",
8   "available": true,
9   "_links": {
10    "self": {
11      "href": "http://localhost:8080/libros/5"
12    }
13  }
14 }

```

<http://localhost:8080/libros>

Se registra un nuevo libro correctamente. El sistema responde con el código 201 Created y devuelve el objeto JSON con los datos del libro añadido.

The screenshot shows a REST client interface with two panels. The top panel shows a successful POST request to `http://localhost:8080/libros` with a status of 201 Created. The request body is a JSON object: `{ "title": "Caperucita Roja", "authors": "Charles Perrault", "edition": "Primera" }`. The bottom panel shows a failed POST request with a status of 400 Bad Request. The response body is a JSON object: `{ "error": "Bad Request", "message": "Validation failed", "errors": { "isbn": "El ISBN no puede estar en blanco", "publisher": "La editorial no puede estar en blanco" }, "timestamp": "2025-06-24T10:19:40.990720100", "status": 400 }`.

```
POST http://localhost:8080/libros

{
  "title": "Caperucita Roja", "authors": "Charles Perrault", "edition": "Primera"
}
```

400 Bad Request • 14 ms • 359 B

```
{
  "error": "Bad Request",
  "message": "Validation failed",
  "errors": {
    "isbn": "El ISBN no puede estar en blanco",
    "publisher": "La editorial no puede estar en blanco"
  },
  "timestamp": "2025-06-24T10:19:40.990720100",
  "status": 400
}
```

<https://localhost:8080/libros>

Como en este caso no están completos todos los campos obligatorios correctamente, el servidor responde 400 Bad Request al fallar la validación y indica los errores detectados.

3. PUT/Libros/{id} :-

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

- Method:** PUT
- URL:** http://localhost:8080/libros/2
- Buttons:** Send, Cookies, Beautify
- Tabs:** Params, Auth, Headers (9), Body (selected), Scripts, Settings
- Body Format:** raw (selected), JSON
- JSON Body:**

```
1 {  
2  
3   "title": "El Quijote Actualizado", "authors": "Miguel de Cervantes",  
4     "edition": "Segunda",  
5   "isbn": "978-3-16-148410-0", "publisher": "Editorial Cervantes", "available":  
6     false  
7 }
```
- Response:** 204 No Content (44 ms, 112 B)
- Response Body:** Raw (selected), Preview, Visualize

<http://localhost:8080/libros/2>

Se actualiza el libro con un identificador específico modificando sus datos. El servidor devuelve dichos datos actualizados incluyendo otros enlaces relacionados.

PUT ▼ http://localhost:8080/libros/999 Send ▼

Params Auth Headers (9) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

```
1 {
2
3   "title": "El Quijote Actualizado", "authors": "Miguel de Cervantes",
4     "edition": "Segunda",
5   "isbn": "978-3-16-148410-0", "publisher": "Editorial Cervantes", "available":
6     false
7 }
```

Body ▼ 🕒 404 Not Found • 20 ms • 130 B • 🌐 ⋮

📄 Raw ▼ ▶ Preview 🔍 Visualize ▼ ☰ 🔍 📄 🔗

1

<http://localhost:8080/libros/999>

La solicitud intenta sin éxito modificar un libro inexistente por su identificador. Devuelve 404 Not Found indicando que no se ha podido realizar la actualización.

4. DELETE/Libros/{id} -

DELETE ▼ http://localhost:8080/libros/5 Send ▼

Params Auth Headers (7) Body ● Scripts Settings Cookies

raw ▼ JSON ▼ Beautify

1 Ctrl+Alt+P for Posthot

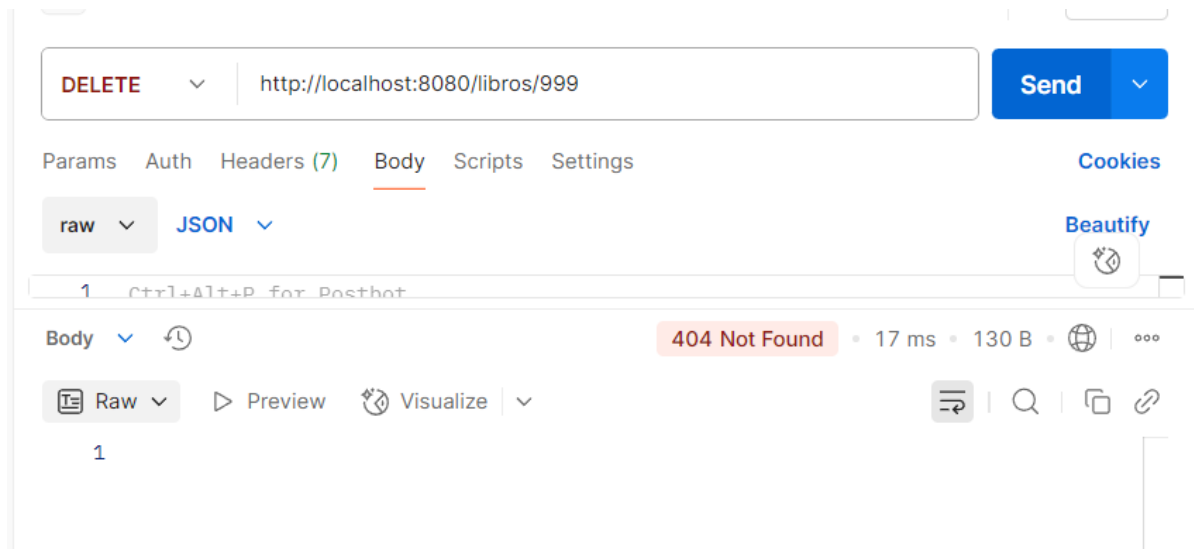
Body ▼ 🕒 204 No Content • 47 ms • 112 B • 🌐 ⋮

📄 Raw ▼ ▶ Preview 🔍 Visualize ▼ ☰ 🔍 📄 🔗

1

<http://localhost:8080/libros/5>

El sistema devuelve 204 Not Content al eliminarse correctamente el libro especificado mediante su identificador. No se devuelven sus datos en la respuesta.



<https://localhost:8080/libros/999>

Como hemos visto anteriormente, la solicitud falla al intentar eliminar un libro que no existe devolviendo el error 404 Not Found.

5.3 Prestamos

1 POST /prestamos

POST

http://localhost:8080/prestamos

Send

Params

Auth

Headers (9)

Body

Scripts

Settings

Cookies

raw

JSON

Beautify

```
1 {
2   "userId": 5,
3   "bookId": 6
4 }
5
6
```

Body

201 Created

79 ms

620 B

{}

JSON

Preview

Visualize

```
1 {
2   "id": 9,
3   "userId": 5,
4   "bookId": 6,
5   "loanDate": "2025-06-27",
6   "dueDate": "2025-07-11",
7   "returnDate": null,
8   "penaltyUntil": null,
9   "_links": {
10    "self": {
11      "href": "http://localhost:8080/prestamos/9"
12    },
13    "estado": {
14      "href": "http://localhost:8080/prestamos/9/estado"
15    },
16    "fecha": {
17      "href": "http://localhost:8080/prestamos/9/fecha"
18    }
19  }
20 }
```

http://localhost:8080/prestamos

Se realiza un préstamo correctamente entre el usuario con id = 5 y el libro con id = 6. El sistema responde con el código 201 Created y devuelve el objeto JSON con los detalles del préstamo generado.

POST http://localhost:8080/prestamos

Params Auth Headers (9) **Body** Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "userId": 999,
3   "bookId": 99
4 }
5
```

Body 404 Not Found • 17 ms • 130 B • Visualize

Raw Preview Visualize

1

http://localhost:8080/prestamos

Se intenta realizar un préstamo con un usuario inexistente. El sistema responde con 404 Not Found al no encontrar el recurso especificado.

POST http://localhost:8080/prestamos

Params Auth Headers (9) **Body** Scripts Settings Cookies Beautify

raw JSON

```
1 {
2   "userId": 6,
3   "bookId": 1
4 }
5
```

Body 403 Forbidden • 14 ms • 130 B • Visualize

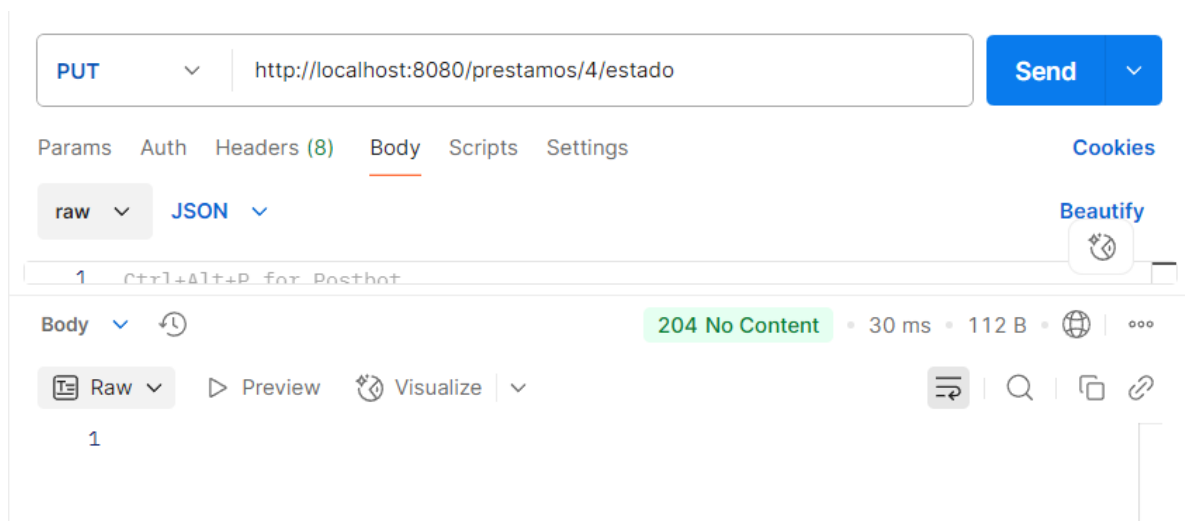
Raw Preview Visualize

1

http://localhost:8080/prestamos

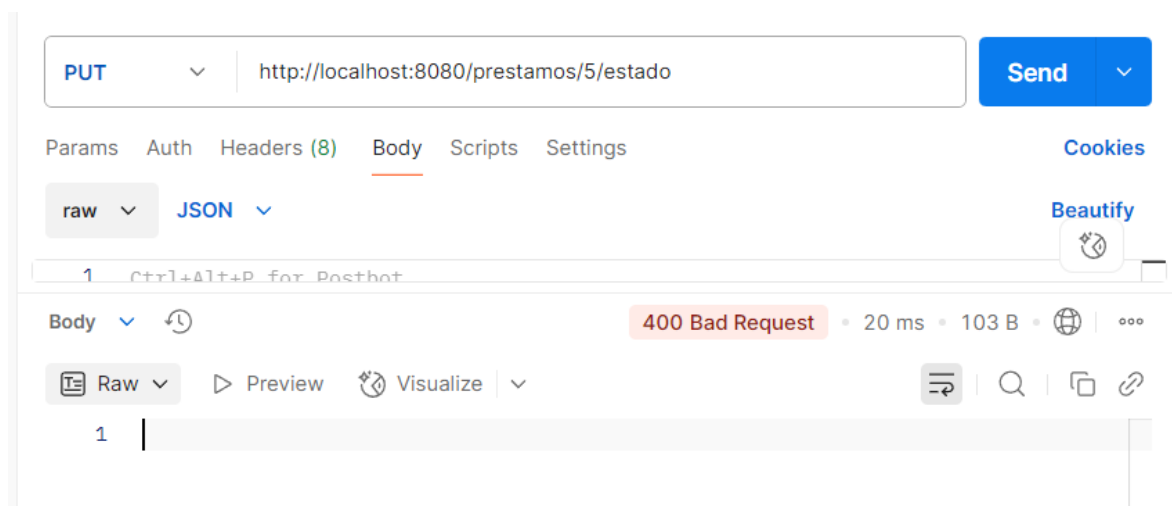
Se intenta hacer un préstamo con un usuario que tiene una penalización activa. El sistema detecta que el usuario con id = 6 fue sancionado por devolver un préstamo con retraso, y responde correctamente con el código 403 Forbidden. Se devuelve un mensaje de error indicando que el usuario no puede realizar préstamos hasta que expire la penalización.

2 PUT /prestamos/{id}/estado



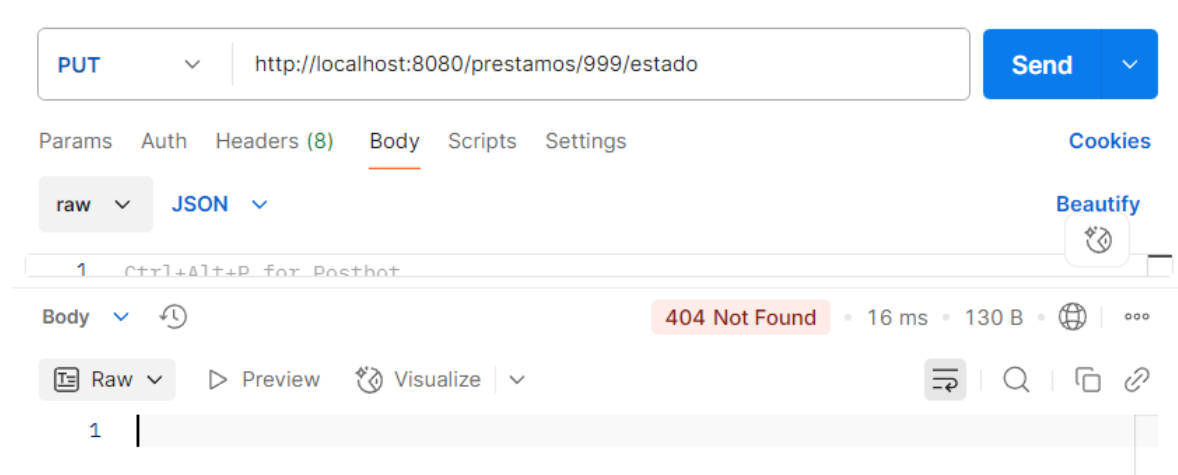
http://localhost:8080/prestamos/4/estado

El préstamo se devuelve correctamente. El sistema detecta que el préstamo con id = 4 estaba activo y lo actualiza correctamente, devolviendo el código 204 No Content.



http://localhost:8080/prestamos/5/estado

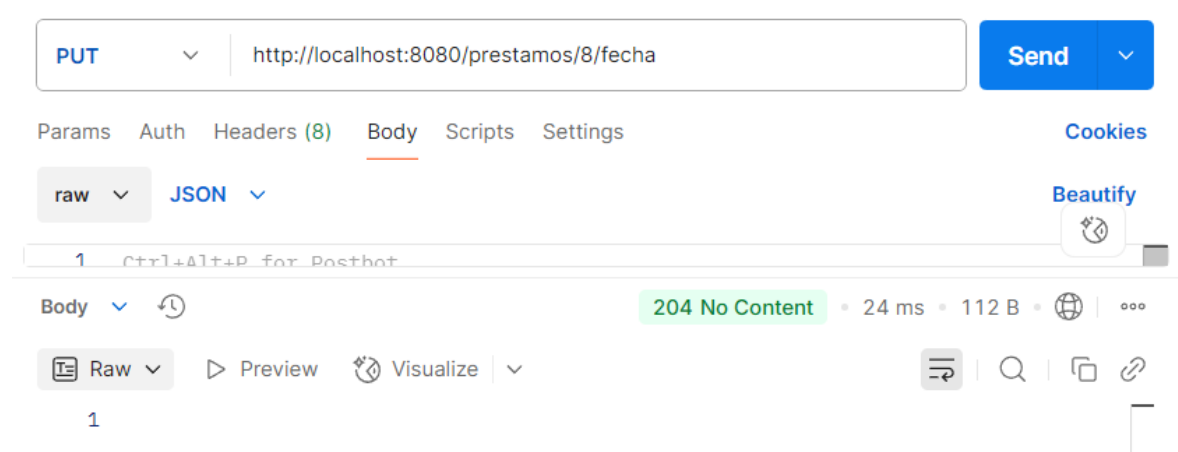
Se intenta devolver un préstamo que ya ha sido devuelto anteriormente. El sistema detecta el error y responde correctamente con el código 400 Bad Request.



http://localhost:8080/prestamos/999/estado

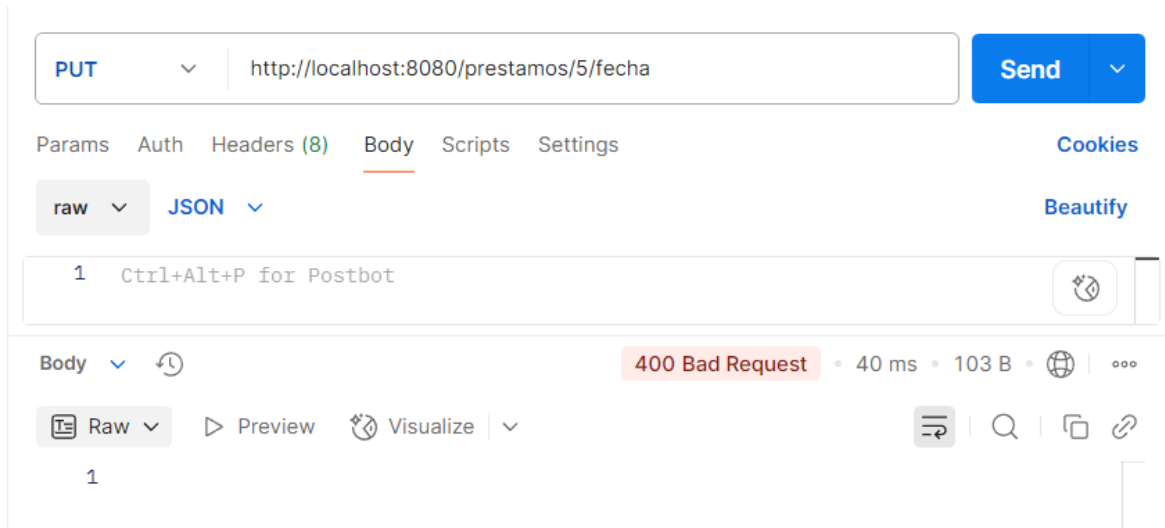
Se intenta devolver un préstamo con un identificador inexistente. El sistema no encuentra el recurso y responde correctamente con el código 404 Not Found.

3 PUT /prestamos/{id}/fecha



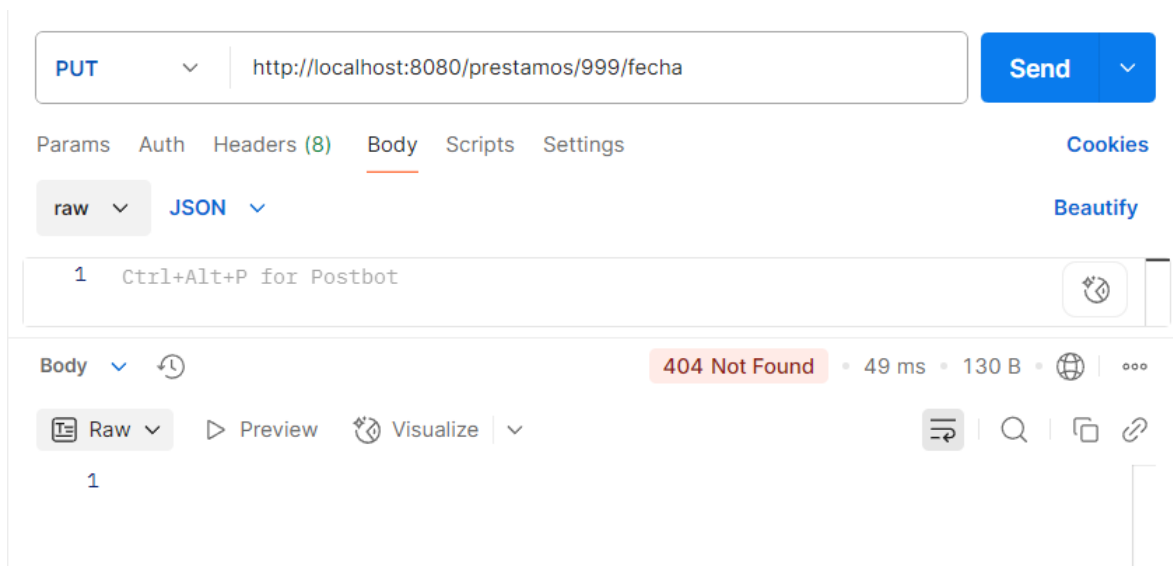
http://localhost:8080/prestamos/8/fecha

El sistema permite extender la fecha de vencimiento de un préstamo activo con identificador 8. La operación se completa correctamente y responde con el código 204 No Content.



http://localhost:8080/prestamos/5/fecha

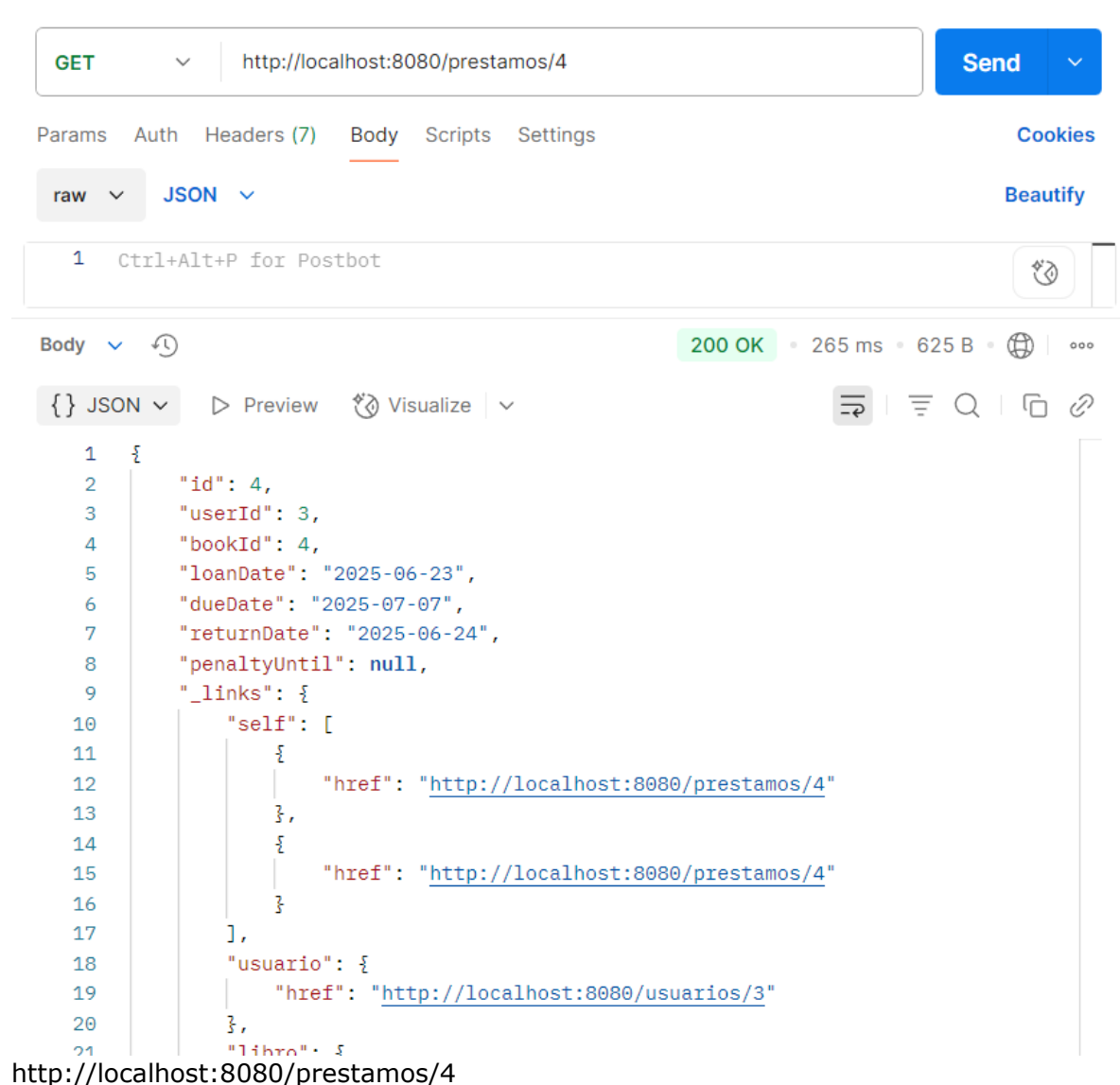
Se intenta extender la fecha de vencimiento de un préstamo ya devuelto. El sistema detecta que la operación no tiene sentido y responde correctamente con el código 400 Bad Request.



http://localhost:8080/prestamos/999/fecha

Se intenta extender un préstamo que no existe. El sistema no encuentra el recurso solicitado y responde correctamente con el código 404 Not Found.

4 GET /prestamos/{id}



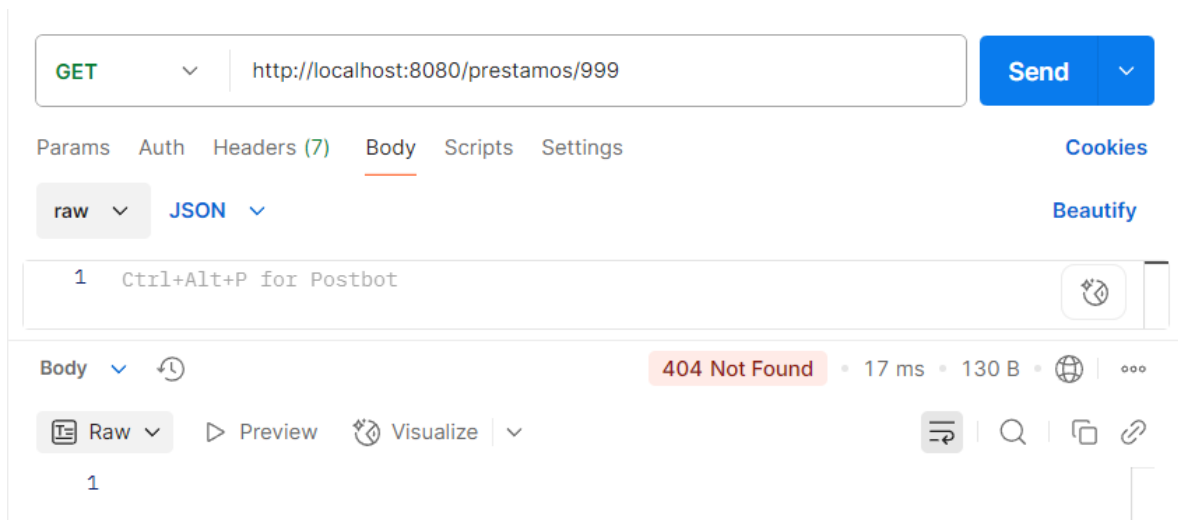
The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:8080/prestamos/4
- Send Button:** A blue button labeled "Send".
- Tabs:** Params, Auth, Headers (7), Body (selected), Scripts, Settings.
- Body Format:** raw (selected), JSON.
- Body Content:** 1 Ctrl+Alt+P for Postbot
- Status Bar:** 200 OK • 265 ms • 625 B • [Globe icon] • [More icon]
- Response Format:** {} JSON (selected), Preview, Visualize.
- JSON Response:**

```
1 {
2   "id": 4,
3   "userId": 3,
4   "bookId": 4,
5   "loanDate": "2025-06-23",
6   "dueDate": "2025-07-07",
7   "returnDate": "2025-06-24",
8   "penaltyUntil": null,
9   "_links": {
10    "self": [
11      {
12        "href": "http://localhost:8080/prestamos/4"
13      },
14      {
15        "href": "http://localhost:8080/prestamos/4"
16      }
17    ],
18    "usuario": {
19      "href": "http://localhost:8080/usuarios/3"
20    },
21    "libro": {
```

http://localhost:8080/prestamos/4

Se consulta un préstamo existente con id = 4. El sistema responde correctamente con el código 200 OK y devuelve la información detallada del préstamo en formato JSON.



http://localhost:8080/prestamos/999

Se intenta consultar un préstamo con un identificador inexistente. El sistema no encuentra el recurso y responde correctamente con el código 404 Not Found.

5 PUT /prestamos

GET <http://localhost:8080/prestamos?startDate=2025-06-01&endDate=2025-06-30> Send

Params Auth Headers (7) **Body** Scripts Settings Cookies

raw JSON Beautify

1 Ctrl+Alt+P for Posthot

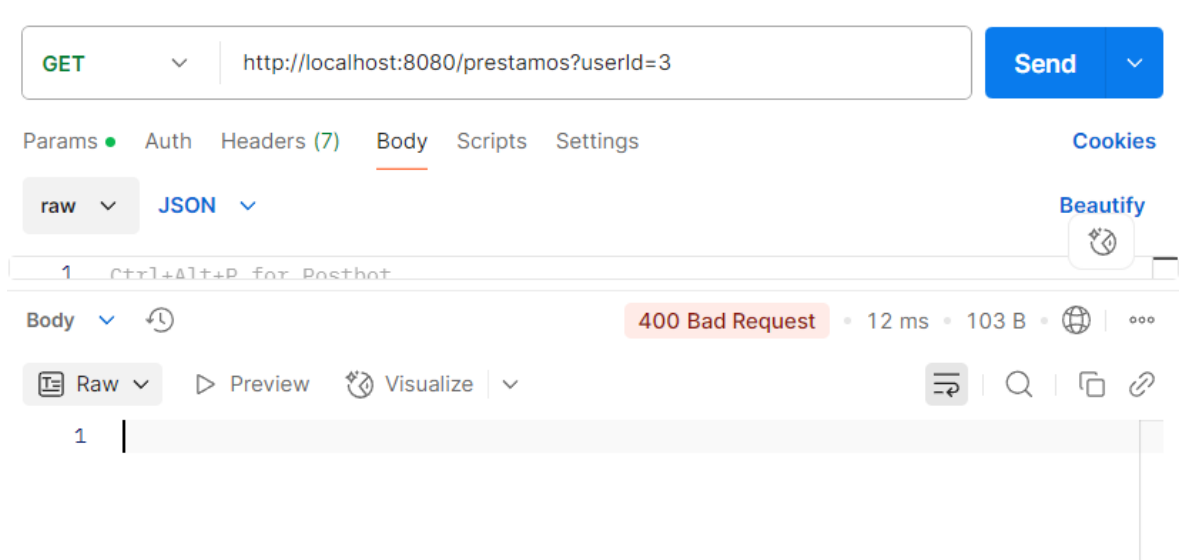
Body 200 OK • 441 ms • 3 KB

```
{
  "_embedded": {
    "prestamoDTOList": [
      {
        "id": 1,
        "userId": null,
        "bookId": 1,
        "loanDate": "2025-04-29",
        "dueDate": "2025-05-13",
        "returnDate": "2025-04-29",
        "penaltyUntil": null,
        "_links": {
          "self": {
            "href": "http://localhost:8080/prestamos/1"
          },
          "usuario": {
            "href": "http://localhost:8080/usuarios/{id}",
            "templated": true
          },
          "libro": {
            "href": "http://localhost:8080/libros/1"
          }
        }
      }
    ]
  }
}
```

<http://localhost:8080/prestamos?startDate=2025-06-01&endDate=2025-06-30>

Se solicita al sistema la lista de préstamos realizados entre el 1 y el 30 de junio de 2025, utilizando parámetros en la cadena de consulta (`startDate` y `endDate`). El sistema responde correctamente con un código 200 OK y devuelve los préstamos registrados en ese intervalo temporal.

<http://localhost:8080/prestamos?userId=3>



En este caso se realiza una consulta para obtener los préstamos del usuario con id = 3. Aunque el usuario existe en el sistema, no tiene préstamos asociados, por lo que el servidor responde con un código 400 Bad Request.

6. CAPTURAS DE LA EJECUCIÓN DEL CLIENTE

En esta sección se documentan las pruebas realizadas con el cliente de consola en Java, desarrollado para interactuar con la API RESTful de gestión de biblioteca.

A diferencia de versiones anteriores del cliente —basadas en menús interactivos—, en esta versión se ha rediseñado por completo su estructura para permitir la automatización de pruebas, agrupando las operaciones en funciones tipo `testX()`. Cada una de estas funciones ejecuta un conjunto de llamadas representativas, simulando distintos casos de uso: creación, modificación, consulta, filtrado, préstamos, devoluciones, etc.

Este enfoque tiene varias ventajas:

- **Facilita la validación** de operaciones encadenadas (por ejemplo, crear un préstamo y luego devolverlo).
- **Permite comprobar la coherencia** del sistema en ejecuciones reales.
- **Agrupar y documentar escenarios típicos**, haciéndolos fácilmente reproducibles.
- Mejora la trazabilidad de errores o inconsistencias, gracias a una salida controlada por consola.

En total se han definido 8 pruebas (`test1()` a `test8()`), ejecutadas secuencialmente. Todas las operaciones se han validado mediante los **códigos HTTP esperados** y la correcta visualización de los datos en consola. A continuación, se explican en detalle las operaciones incluidas en cada test

6.1 Test 1 - Crear usuario, libro y préstamo (devolución)

Este primer test valida el flujo más elemental del sistema: registrar un nuevo usuario, añadir un libro, realizar un préstamo y efectuar su devolución. El objetivo es asegurar que las operaciones básicas del servicio REST funcionan correctamente y de forma encadenada, conforme a lo especificado.

Las operaciones realizadas son las siguientes:

1. **Creación de usuario** (POST /usuarios): se envían los datos de un nuevo usuario.
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** el servidor responde con el identificador asignado al nuevo usuario.
2. **Creación de libro** (POST /libros): se registra un nuevo libro en el sistema.
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se obtiene el identificador del libro recién creado.
3. **Creación de préstamo** (POST /prestamos): se asocia el libro al usuario mediante un préstamo.
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se devuelve el identificador del préstamo registrado.
4. **Devolución del préstamo** (PUT /prestamos/{id}/estado): se actualiza el estado del préstamo para marcarlo como devuelto.
 - **Código HTTP esperado:** 204 No Content
 - **Resultado:** la respuesta es vacía, lo que indica una actualización exitosa sin contenido de retorno.

```
===== INICIO TEST 1 =====  
[crearUsuario] Código: 201  
[crearUsuario] ID: 9  
[crearLibro] Código: 201  
[crearLibro] ID: 13  
[crearPrestamo] Código: 201  
[crearPrestamo] ID: 9  
[devolverPrestamo] Código: 204  
[devolverPrestamo] Respuesta:  
===== FIN TEST 1 =====
```


6.2 Test 2 - Usuario Básico: creación, consulta individual y listado general

Este test tiene como objetivo verificar las funcionalidades relacionadas con la gestión de usuarios desde un punto de vista básico. Se evalúa si el sistema permite registrar un nuevo usuario, consultarlo por su identificador, y obtener la lista completa de usuarios registrados, todo ello utilizando los endpoints REST definidos.

Las operaciones realizadas son:

1. **Creación de usuario** (POST /usuarios):
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** el sistema devuelve el ID del nuevo usuario creado (juan_test).
2. **Consulta individual del usuario** (GET /usuarios/{id}):
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se muestran los datos personales del usuario registrado.
3. **Listado de usuarios** (GET /usuarios):
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se devuelve una lista paginada con los usuarios registrados en la base de datos, incluyendo al nuevo usuario creado.

Este test confirma el correcto funcionamiento del CRUD de usuarios en sus operaciones básicas de alta y consulta.

```
===== INICIO TEST 2: Usuario Básico =====
[crearUsuario] Código: 201
[crearUsuario] ID: 10

--> Consultar Usuario por ID: 10
[consultarUsuario] Código: 200
Usuario ID: 10 | Nombre: juan_test | Matrícula: AA001 | Fecha Nacimiento: 1995-05-05 | Email: juan@test.com
[listarUsuarios] Código: 200
Usuario ID: 1 | Nombre: juanperez | Matrícula: A001 | Nacimiento: 1990-05-15 | Email: juan@example.com
Usuario ID: 2 | Nombre: mariafernandez | Matrícula: A002 | Nacimiento: 1988-07-21 | Email: maria@example.com
Usuario ID: 3 | Nombre: carlossoto | Matrícula: A003 | Nacimiento: 1995-02-10 | Email: carlos@example.com
Usuario ID: 4 | Nombre: lauralopez | Matrícula: A004 | Nacimiento: 2000-11-01 | Email: laura@example.com
Usuario ID: 5 | Nombre: alfonso | Matrícula: A005 | Nacimiento: 1999-06-15 | Email: alfonso@example.com
Usuario ID: 6 | Nombre: alfon | Matrícula: d210253 | Nacimiento: 2003-03-22 | Email: a.marinmite
Usuario ID: 7 | Nombre: pepe_test | Matrícula: 1234567 | Nacimiento: 2000-01-01 | Email: pepe@test.com
Usuario ID: 8 | Nombre: ana_modificada | Matrícula: 9999999 | Nacimiento: 1990-01-01 | Email: ana@nueva.com
Usuario ID: 9 | Nombre: carlos_test | Matrícula: 934719 | Nacimiento: 2004-03-24 | Email: carlos@test.com
Usuario ID: 10 | Nombre: juan_test | Matrícula: AA001 | Nacimiento: 1995-05-05 | Email: juan@test.com
===== FIN TEST 2 =====
```

6.3 Test 3 - Préstamo Básico: creación y consulta de préstamos activos

Este test valida el flujo mínimo necesario para realizar un préstamo y consultarlo como activo. Se comprueba que el sistema permite asociar correctamente un usuario con un libro disponible, y posteriormente, recuperar la información del préstamo desde el punto de vista del usuario.

Las operaciones realizadas son:

1. **Creación de usuario** (POST /usuarios)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se crea un nuevo usuario (maria_test).
2. **Alta de libro disponible** (POST /libros)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se registra un nuevo libro con el campo available=true.
3. **Creación de préstamo** (POST /prestamos)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se genera un préstamo correctamente asociado al usuario y al libro.
4. **Consulta de préstamos activos del usuario** (GET /usuarios/{id}/prestamos)
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se obtiene la lista de préstamos sin returnDate, en este caso, uno solo. Se muestra su fecha de inicio, de vencimiento y el estado "Devuelto: No".

Este test verifica correctamente que se puede generar un préstamo válido y recuperarlo como préstamo activo asociado a un usuario.

```
===== INICIO TEST 3: Préstamo Básico =====
[crearUsuario] Código: 201
[crearUsuario] ID: 11
[crearLibro] Código: 201
[crearLibro] ID: 14
[crearPrestamo] Código: 201
[crearPrestamo] ID: 10

--> Consultar Préstamos Activos de Usuario ID: 11
[consultarPrestamosActivos] Código: 200
Préstamo ID: 10 | Fecha préstamo: 2025-06-28 | Vence: 2025-07-12 | Devuelto: No
===== FIN TEST 3 =====
```

6.4 Test 4 - Ampliar y Devolver Préstamo

Este test pone a prueba el ciclo completo de vida de un préstamo: creación, ampliación de plazo, devolución, y consulta posterior tanto del historial como de la actividad completa del usuario. Se confirma así que los cambios de estado del préstamo son correctamente registrados y reflejados en las distintas vistas del sistema.

Las operaciones realizadas son:

1. **Creación de usuario** (POST /usuarios)
 - **Código HTTP esperado:** 201 Created
 - Resultado: se da de alta a un nuevo usuario (ana_test).
2. **Alta de libro disponible** (POST /libros)
 - **Código HTTP esperado:** 201 Created
 - Resultado: se registra un libro disponible para préstamo.
3. **Creación del préstamo** (POST /prestamos)
 - **Código HTTP esperado:** 201 Created
 - Resultado: se genera un préstamo activo asociado al usuario y libro creados.
4. **Ampliación del préstamo** (PUT /prestamos/{id}/fecha)
 - **Código HTTP esperado:** 204 No Content
 - Resultado: se extiende correctamente la fecha de vencimiento del préstamo. En el resultado se puede observar que la fecha de vencimiento fue incrementada.
5. **Devolución del préstamo** (PUT /prestamos/{id}/estado)
 - **Código HTTP esperado:** 204 No Content
 - Resultado: el préstamo es marcado como devuelto correctamente.
6. **Consulta del historial del usuario** (GET /usuarios/{id}/historial)
 - **Código HTTP esperado:** 200 OK
 - Resultado: se observa el préstamo devuelto, incluyendo su fecha de devolución.
7. **Consulta de actividad completa del usuario** (GET /usuarios/{id}/actividad)
 - **Código HTTP esperado:** 200 OK
 - Resultado: muestra la ficha del usuario, sin préstamos activos, y con el préstamo recién devuelto entre los últimos cinco préstamos finalizados.

Este test verifica el correcto funcionamiento de las operaciones de ampliación y devolución, así como la consistencia entre los distintos puntos de consulta de préstamos.

```

===== INICIO TEST 4: Ampliar y Devolver Préstamo =====
[crearUsuario] Código: 201
[crearUsuario] ID: 12
[crearLibro] Código: 201
[crearLibro] ID: 15
[crearPrestamo] Código: 201
[crearPrestamo] ID: 11

--- Ampliando préstamo ---
[ampliarPrestamo] Código: 204
[ampliarPrestamo] Respuesta:

--- Devolviendo préstamo ---
[devolverPrestamo] Código: 204
[devolverPrestamo] Respuesta:

--- Consultando historial ---

--> Consultar Historial de Préstamos del Usuario ID: 12
[consultarHistorialPrestamos] Código: 200
Préstamo ID: 11 | Préstamo: 2025-06-28 | Vence: 2025-07-26 | Devolución: 2025-06-28

--- Consultando actividad completa ---

--> Consultar Actividad Completa del Usuario ID: 12
[consultarActividadUsuario] Código: 200

=== Datos del Usuario ===
Nombre: ana_test | Matrícula: ZZ999 | Nacimiento: 1985-03-15 | Email: ana@test.com

=== Préstamos Activos ===

=== Últimos 5 Préstamos Devueltos ===
Préstamo ID: 11 | Préstamo: 2025-06-28 | Vence: 2025-07-26 | Devuelto: 2025-06-28
===== FIN TEST 4 =====

```

6.5 Test 5 - Filtrado por título + disponibilidad

Este test tiene como objetivo verificar el funcionamiento del filtrado combinado por **patrón de título** y **disponibilidad** en la operación de listado de libros. Se comprueba que el sistema responde correctamente ante búsquedas de libros cuyo título contiene una palabra concreta, y permite limitar el resultado a los que estén disponibles para préstamo. Además, se realiza un préstamo intermedio para observar cómo cambia la disponibilidad de un libro en el listado.

Las operaciones realizadas son:

1. **Listado por título (GET /libros?titulo=Progra)**

Se listan todos los libros cuyo título contiene la palabra "Progra", sin filtrar por disponibilidad.

- a. **Código HTTP esperado:** 200 OK
- b. **Resultado:** se devuelven 3 libros con ese patrón en el título, independientemente de si están disponibles o no.

2. **Listado por título y disponibles (GET /libros?titulo=Progra&disponible=true)**

Se listan solo los libros cuyo título contiene "Progra" y estén disponibles.

- a. **Código HTTP esperado:** 200 OK
- b. **Resultado:** se devuelven únicamente aquellos libros del paso anterior que tengan available=true.

3. **Préstamo de uno de los libros (POST /prestamos)**

Se realiza un préstamo del libro "Programación en Java (Edición 3)", que estaba disponible.

- a. **Código HTTP esperado:** 201 Created
- b. **Resultado:** el libro pasa a estar no disponible tras asociarse a un préstamo.

4. **Listado por título tras préstamo (GET /libros?titulo=Progra)**

Se repite el listado por patrón "Progra", sin filtrar por disponibilidad.

- a. **Código HTTP esperado:** 200 OK
- b. **Resultado:** se muestran los 3 libros como en el paso 1, pero ahora con uno de ellos marcado como Disponible: false.

5. **Listado por título y disponibles tras préstamo (GET /libros?titulo=Progra&disponible=true)**

Se vuelve a listar por patrón "Progra", esta vez filtrando por disponibilidad.

- a. **Código HTTP esperado:** 200 OK
- b. **Resultado:** ya no aparece el libro prestado, confirmando que ha pasado a no estar disponible.

```
===== INICIO TEST 5: Filtrado por título + disponibilidad =====

--- Libros con 'Progra' en el título (todos) ---

--> Listar Libros
[listarLibros] Código: 200
Libro ID: 8 | Título: Programación en Java | Autores: Deitel & Deitel | Edición: 1 | ISBN: 978-916482645 | Disponible: false}
Libro ID: 9 | Título: Programación en Java (Edición 2) | Autores: Deitel & Deitel | Edición: 2 | ISBN: 978-015493325 | Disponible: true}
Libro ID: 10 | Título: Programación en Java (Edición 3) | Autores: Deitel & Deitel | Edición: 3 | ISBN: 978-657399031 | Disponible: true}}

--- Libros con 'Progra' en el título (solo disponibles) ---

--> Listar Libros
[listarLibros] Código: 200
Libro ID: 9 | Título: Programación en Java (Edición 2) | Autores: Deitel & Deitel | Edición: 2 | ISBN: 978-015493325 | Disponible: true}
Libro ID: 10 | Título: Programación en Java (Edición 3) | Autores: Deitel & Deitel | Edición: 3 | ISBN: 978-657399031 | Disponible: true}}

--- Realizando préstamo del libro ID 10 ---
[crearPrestamo] Código: 201
[crearPrestamo] ID: 12

--- Libros con 'Progra' en el título tras préstamo (todos) ---

--> Listar Libros
[listarLibros] Código: 200
Libro ID: 8 | Título: Programación en Java | Autores: Deitel & Deitel | Edición: 1 | ISBN: 978-916482645 | Disponible: false}
Libro ID: 9 | Título: Programación en Java (Edición 2) | Autores: Deitel & Deitel | Edición: 2 | ISBN: 978-015493325 | Disponible: true}
Libro ID: 10 | Título: Programación en Java (Edición 3) | Autores: Deitel & Deitel | Edición: 3 | ISBN: 978-657399031 | Disponible: false}}

--- Libros con 'Progra' en el título tras préstamo (solo disponibles) ---

--> Listar Libros
[listarLibros] Código: 200
Libro ID: 9 | Título: Programación en Java (Edición 2) | Autores: Deitel & Deitel | Edición: 2 | ISBN: 978-015493325 | Disponible: true}}
===== FIN TEST 5 =====
```

6.6 Test 6 - Consultar Préstamos Activos e Historial

Este test verifica que el sistema permite recuperar correctamente tanto los **préstamos activos** como el **historial completo** de préstamos de un usuario. Se simula el caso de un usuario que tiene varios préstamos, de los cuales uno se devuelve y otro queda activo. A continuación, se consultan las vistas correspondientes del sistema.

Las operaciones realizadas son:

1. **Creación de usuario** (POST /usuarios)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se crea un nuevo usuario llamado mario_test.
2. **Alta de libros** (POST /libros)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se registran dos libros disponibles para préstamo: "Libro Mario 1" y "Libro Mario 2".
3. **Creación de préstamos** (POST /prestamos)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se crean dos préstamos vinculando ambos libros con el usuario.
4. **Devolución de un préstamo** (PUT /prestamos/{id}/estado)
 - **Código HTTP esperado:** 204 No Content
 - **Resultado:** uno de los préstamos queda marcado como devuelto, actualizando su estado correctamente.
5. **Consulta de préstamos activos del usuario** (GET /usuarios/{id}/prestamos)
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se devuelve únicamente el préstamo activo (aquel que aún no tiene returnDate).
6. **Consulta del historial completo de préstamos del usuario** (GET /usuarios/{id}/historial)
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se muestra correctamente el préstamo devuelto, con fechas de préstamo, vencimiento y devolución.

Este test garantiza que las vistas del sistema distinguen correctamente entre préstamos activos y préstamos pasados del usuario, reflejando de forma precisa su historial.

```
===== INICIO TEST 6: Consultar Préstamos Activos e Historial =====
[crearUsuario] Código: 201
[crearUsuario] ID: 13
[crearLibro] Código: 201
[crearLibro] ID: 16
[crearLibro] Código: 201
[crearLibro] ID: 17
[crearPrestamo] Código: 201
[crearPrestamo] ID: 13
[crearPrestamo] Código: 201
[crearPrestamo] ID: 14
[devolverPrestamo] Código: 204
[devolverPrestamo] Respuesta:

--- Préstamos activos del usuario ---

--> Consultar Préstamos Activos de Usuario ID: 13
[consultarPrestamosActivos] Código: 200
Préstamo ID: 13 | Fecha préstamo: 2025-06-28 | Vence: 2025-07-12 | Devuelto: No

--- Historial completo del usuario ---

--> Consultar Historial de Préstamos del Usuario ID: 13
[consultarHistorialPrestamos] Código: 200
Préstamo ID: 14 | Préstamo: 2025-06-28 | Vence: 2025-07-12 | Devolución: 2025-06-28
===== FIN TEST 6 =====
```


6.7 Test 7 - Ampliar Préstamo

Este test verifica que el sistema permite **extender el plazo de vencimiento** de un préstamo activo. Se realiza una ampliación del préstamo y se comprueba que la fecha de vencimiento se ha actualizado correctamente. Además, se valida que el estado del préstamo permanece como activo.

Las operaciones realizadas son:

1. **Creación de usuario** (POST /usuarios)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se crea un nuevo usuario llamado laura_test.
2. **Alta de libro disponible** (POST /libros)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se registra un nuevo libro con disponibilidad para préstamo.
3. **Creación del préstamo** (POST /prestamos)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se asocia el libro al usuario mediante un préstamo activo.
4. **Consulta previa del préstamo activo** (GET /usuarios/{id}/prestamos)
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se muestra la fecha de vencimiento original (por defecto, 14 días después del préstamo).
5. **Ampliación del préstamo** (PUT /prestamos/{id}/fecha)
 - **Código HTTP esperado:** 204 No Content
 - **Resultado:** el préstamo se amplía correctamente y la fecha de vencimiento se incrementa en 14 días adicionales.
6. **Consulta posterior del préstamo activo** (GET /usuarios/{id}/prestamos)
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se observa que la nueva fecha de vencimiento ha sido actualizada correctamente, mientras el estado del préstamo continúa como no devuelto.

Este test comprueba que el endpoint de ampliación de préstamo funciona conforme a lo esperado y que los cambios se reflejan de inmediato en las consultas activas.

```
===== INICIO TEST 7: Ampliar Préstamo =====
[crearUsuario] Código: 201
[crearUsuario] ID: 14
[crearLibro] Código: 201
[crearLibro] ID: 18
[crearPrestamo] Código: 201
[crearPrestamo] ID: 15

--- Antes de la ampliación ---

--> Consultar Préstamos Activos de Usuario ID: 14
[consultarPrestamosActivos] Código: 200
Préstamo ID: 15 | Fecha préstamo: 2025-06-28 | Vence: 2025-07-12 | Devuelto: No

--- Ampliando préstamo ID: 15 ---
[ampliarPrestamo] Código: 204
[ampliarPrestamo] Respuesta:

--- Después de la ampliación ---

--> Consultar Préstamos Activos de Usuario ID: 14
[consultarPrestamosActivos] Código: 200
Préstamo ID: 15 | Fecha préstamo: 2025-06-28 | Vence: 2025-07-26 | Devuelto: No
===== FIN TEST 7 =====
```

6.8 Test 8 - Consultar Actividad de Usuario

Este test evalúa el comportamiento del endpoint que permite **consultar toda la actividad asociada a un usuario**, incluyendo información personal, préstamos activos y los últimos préstamos devueltos. Se simula un escenario en el que el usuario tiene varios préstamos registrados, algunos de ellos devueltos y otros aún activos.

Las operaciones realizadas son:

1. **Creación de usuario** (POST /usuarios)
 - **Código HTTP esperado:** 201 Created
 - **Resultado:** se crea un nuevo usuario identificado como ana_test.
2. **Alta de libros disponibles** (POST /libros)
 - **Código HTTP esperado:** 201 Created (x3)
 - **Resultado:** se registran tres libros disponibles para préstamo.
3. **Creación de préstamos** (POST /prestamos)
 - **Código HTTP esperado:** 201 Created (x3)
 - **Resultado:** se generan tres préstamos asociados al usuario y a los libros creados.
4. **Devolución de préstamos** (PUT /prestamos/{id}/estado)
 - **Código HTTP esperado:** 204 No Content (x2)
 - **Resultado:** dos de los tres préstamos son marcados como devueltos correctamente.
5. **Consulta de actividad completa** (GET /usuarios/{id}/estado)
 - **Código HTTP esperado:** 200 OK
 - **Resultado:** se devuelve la información del usuario, incluyendo:
 - i. Datos personales del usuario
 - ii. Préstamos actualmente activos
 - iii. Historial de los últimos 5 préstamos devueltos (ordenados por fecha de devolución)

Este test verifica la correcta agregación de datos en el resumen de actividad del usuario y confirma que los estados de los préstamos se reflejan adecuadamente.

===== INICIO TEST 8: Consultar Actividad de Usuario =====

[crearUsuario] Código: 201

[crearUsuario] ID: 15

[crearLibro] Código: 201

[crearLibro] ID: 19

[crearLibro] Código: 201

[crearLibro] ID: 20

[crearLibro] Código: 201

[crearLibro] ID: 21

[crearPrestamo] Código: 201

[crearPrestamo] ID: 16

[crearPrestamo] Código: 201

[crearPrestamo] ID: 17

[crearPrestamo] Código: 201

[crearPrestamo] ID: 18

[devolverPrestamo] Código: 204

[devolverPrestamo] Respuesta:

[devolverPrestamo] Código: 204

[devolverPrestamo] Respuesta:

--> Consultar Actividad Completa del Usuario ID: 15

[consultarActividadUsuario] Código: 200

=== Datos del Usuario ===

Nombre: ana_test | Matrícula: A999 | Nacimiento: 1988-08-08 | Email: ana@test.com

=== Préstamos Activos ===

Préstamo ID: 18 | Préstamo: 2025-06-28 | Vence: 2025-07-12 | Devuelto: No

=== Últimos 5 Préstamos Devueltos ===

Préstamo ID: 16 | Préstamo: 2025-06-28 | Vence: 2025-07-12 | Devuelto: 2025-06-28

Préstamo ID: 17 | Préstamo: 2025-06-28 | Vence: 2025-07-12 | Devuelto: 2025-06-28

===== FIN TEST 8 =====

7. REFERENCIAS

[1] Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. University of California, Irvine.

[2] RESTful Web Services. *RESTful Web Services Tutorial*. Disponible en: <https://restfulapi.net/>

[3] Spring Framework. *Spring Boot Documentation*. Disponible en: <https://spring.io/projects/spring-boot>