Library Management System Documentation

Project Description

The Library Management System API is designed to facilitate the management of books, patrons, and borrowing records for a library. This application is built using Spring Boot and provides a RESTful API to manage the various aspects of a library's operations.

Getting Started

Prerequisites

- **Java Development Kit (JDK)**: Ensure that JDK 11 or higher is installed.
- **Maven**: Make sure Maven is installed for dependency management and build tasks.
- **PostgreSQL**: The project uses PostgreSQL for database management.
- **Intellij ultimate IDE**
- **Postman for testing**

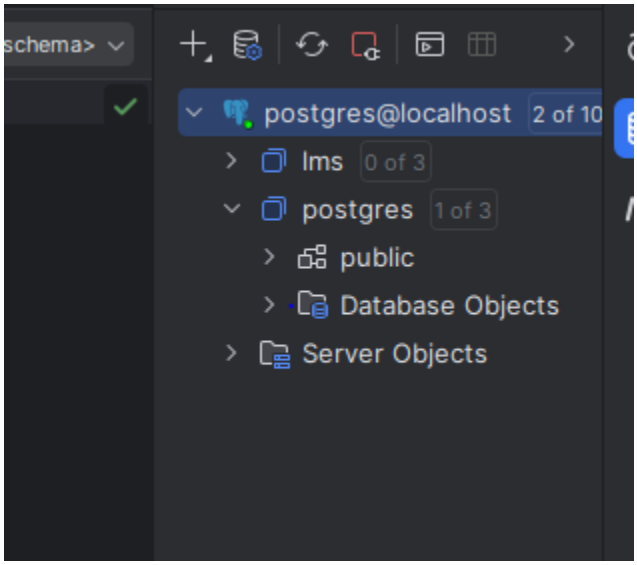
Running the Application

1. **Clone the Repository: Clone the project repository to your local machine.**

   git clone https://github.com/robamaged/LibraryManagementSystem.git

2. **Open in IntelliJ IDEA Ultimate: Import the project into IntelliJ IDEA Ultimate.**
3. **Configure PostgreSQL: Set up PostgreSQL on your local machine and update the database configuration in the `application.yml` file.**
   + -> Datasource -> postreSQL

Create ->new ->database -> write database name “lms”



2 of 10 -> check lms

0 of 3 -> check all schemas under lms database

```yaml
server:
  port: 8090
spring:
  application:
    name: users
  config:
    import: optional:configserver:http://localhost:8888
  datasource:
    driver-class-name: org.postgresql.Driver
    url: jdbc:postgresql://localhost:5432/lms
    username: postgres
    password: 1651
  jpa:
    hibernate:
      ddl-auto: create-drop
    show_sql: true
    database: postgresql
    database-platform: org.hibernate.dialect.PostgreSQLDialect
```

In url type the name of the database created in previous step “lms” ,and change Username and password with your username and password of postgres

4. **Build and Run: Use Maven to build the project, and then run the application within IntelliJ IDEA.**

Interacting with API Endpoints using postman:

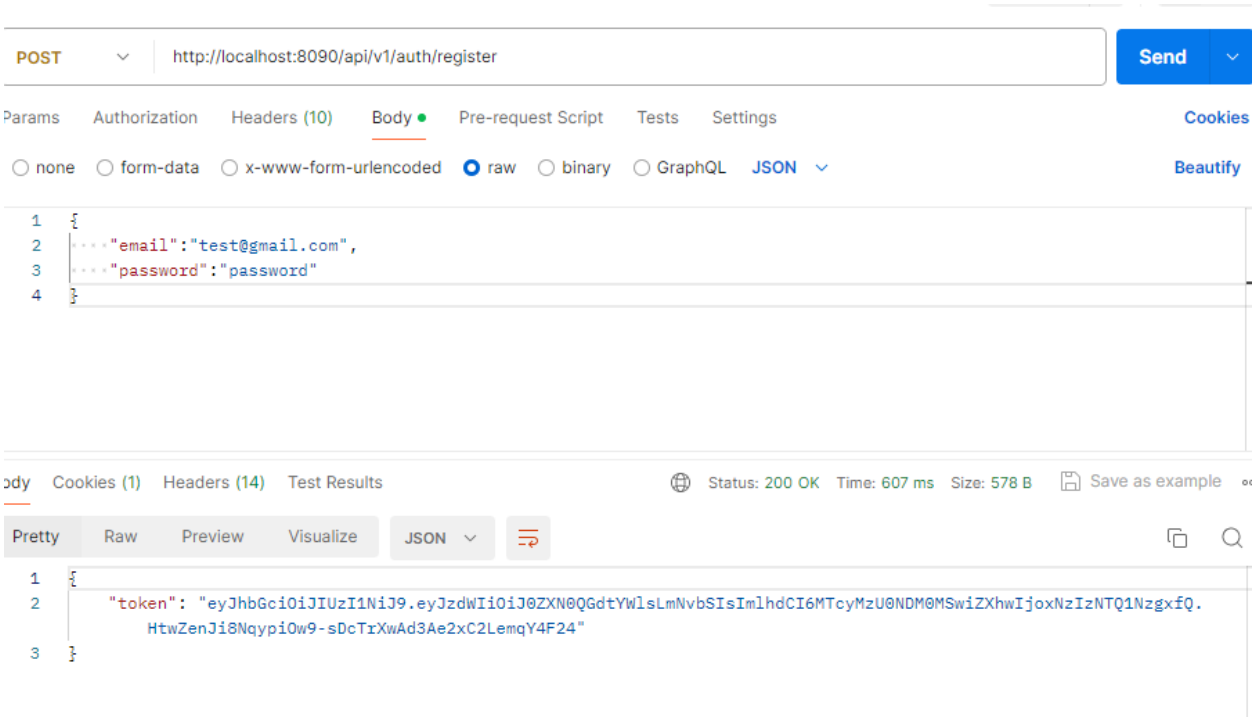
1. Register http://localhost:8090/api/v1/auth/register

- **Request Body:** {

```
    "email":"test@gmail.com",

    "Password":"password"

}
```



Copy the token(needed for later use).

2. Add a New Book

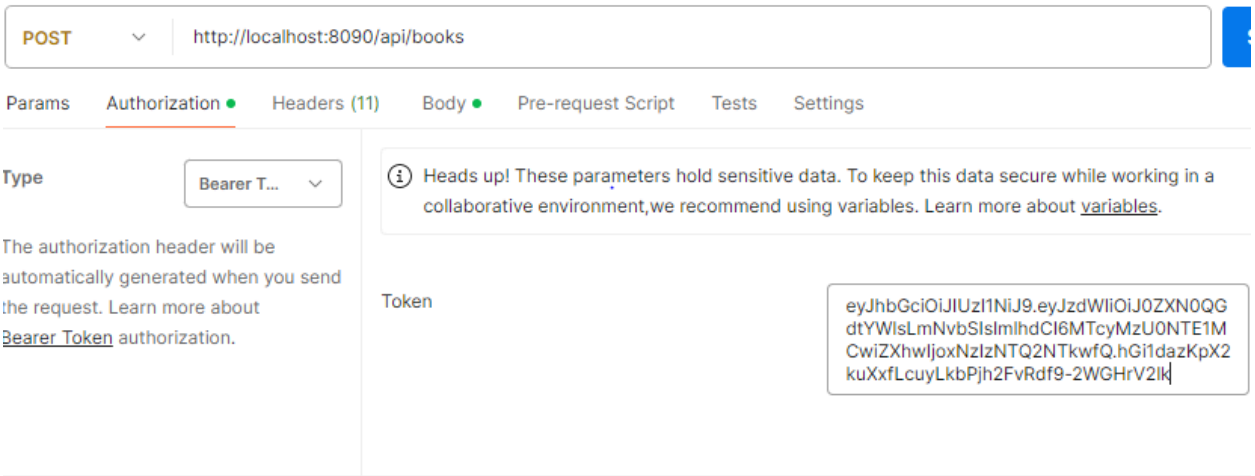
Endpoint:

- **Method:** `POST /api/books`

Authorization tab-> Type:Bearer Token -> paste token to allow for endpoint access

- **Request Body** :{

```
  "title": "Effective Java",
  "author": "Joshua Bloch",
  "publicationYear": 2008,
  "isbn": "9780134685991"
}
```



POST http://localhost:8090/api/books

Params Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "title": "Effective Java",
  "author": "Joshua Bloch",
  "publicationYear": 2008,
  "isbn": "9780134685991"
}
```

Body Cookies (1) Headers (14) Test Results Status: 200 OK Time: 156 ms Size: 526 B

Pretty Raw Preview Visualize JSON

```
{
    "id": 1,
    "title": "Effective Java",
    "author": "Joshua Bloch",
    "publicationYear": 2008,
    "isbn": "9780134685991"
}
```

3. Add a New Patron http://localhost:8090/api/patrons

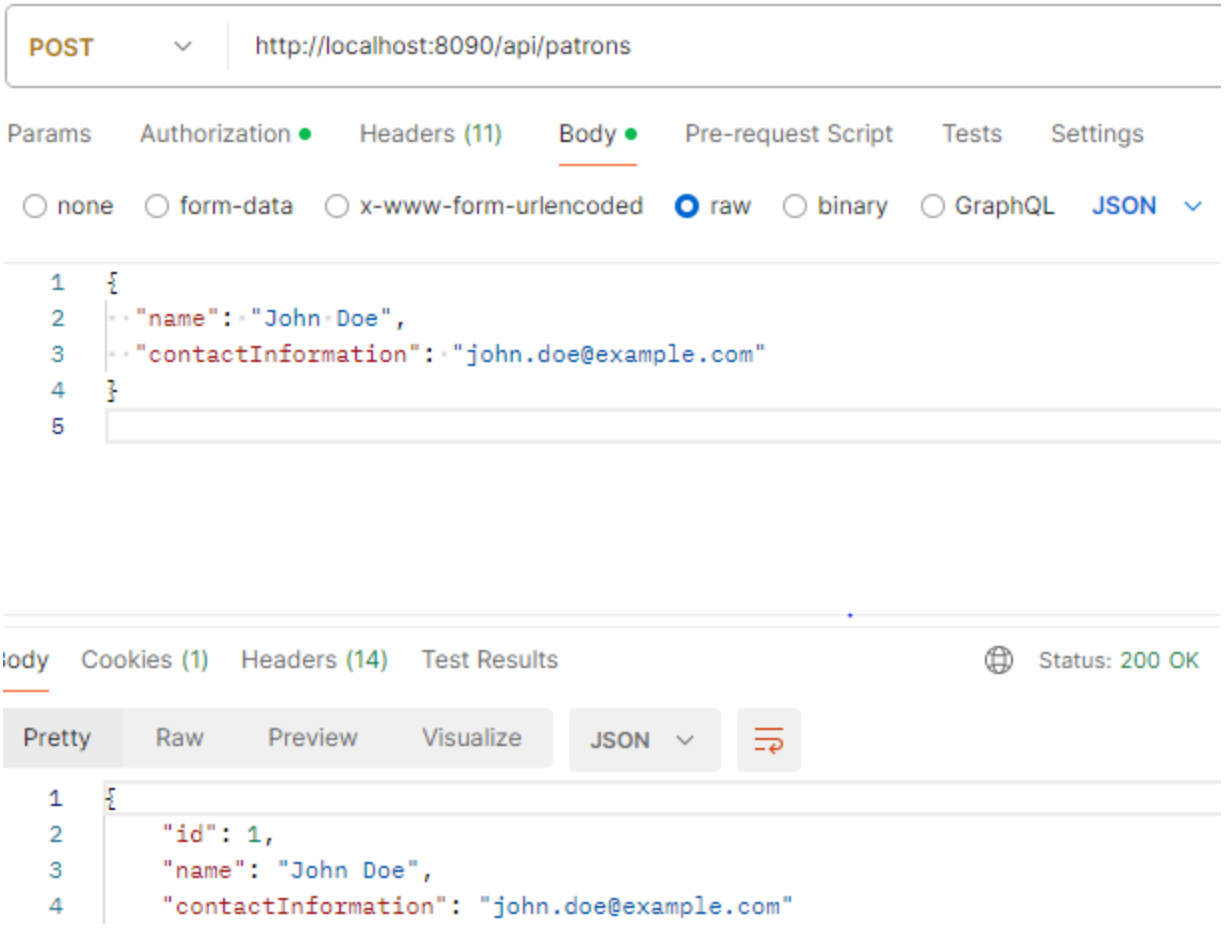
Endpoint:

- **Method:** `POST /api/patrons`
- **Request Body:{**

  "name": "John Doe",

  "contactInformation": "john.doe@example.com"

  }



3. Retrieve All Books

Endpoint:

- **Method:** `GET /api/books`

GET http://localhost:8090/api/books

Params Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
{
  "name": "John Doe",
  "contactInformation": "john.doe@example.com"
}
```

ody Cookies (1) Headers (14) Test Results Status: 200 OK Time: 47 ms

Pretty Raw Preview Visualize JSON

```
[
    {
        "id": 1,
        "title": "Effective Java",
        "author": "Joshua Bloch",
        "publicationYear": 2008,
        "isbn": "9780134685991"
    }
]
```

4. Retrieve All Patrons

Endpoint:

- **Method:** `GET /api/patrons`
- **Expected Response:**

GET http://localhost:8090/api/patrons

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1
```

ody Cookies (1) Headers (14) Test Results Status: 200 OK Time: 25 ms

Pretty Raw Preview Visualize JSON

```
1 [
2     {
3         "id": 1,
4         "name": "John Doe",
5         "contactInformation": "john.doe@example.com"
6     }
7 ]
```

5. Borrow a Book

Endpoint:

- **Method:** `POST /api/borrow/{bookId}/patron/{patronId}`
- **URL:** `/api/borrow/1/patron/1`
- **Expected Response:**

POST http://localhost:8090/api/borrow/1/patron/1

Params Authorization Headers (10) Body Pre-request Script Tests Settings

ody Cookies (1) Headers (14) Test Results Status: 201 Created

Pretty Raw Preview Visualize JSON


```
{
    "id": 1,
    "book": {
        "id": 1,
        "title": "Effective Java",
        "author": "Joshua Bloch",
        "publicationYear": 2008,
        "isbn": "9780134685991"
    },
    "patron": {
        "id": 1,
        "name": "John Doe",
        "contactInformation": "john.doe@example.com"
    },
    "borrowDate": "2024-08-13",
    "returnDate": null
}
```


Endpoint:

- **Method:** `PUT /api/return/{bookId}/patron/{patronId}`
- **URL:** `/api/return/1/patron/1`
- **Expected Response:**

PUT http://localhost:8090/api/return/1/patron/1

Params Authorization Headers (10) Body Pre-request Script Tests Settings

ody Cookies (1) Headers (14) Test Results Status: 200 OK

Pretty Raw Preview Visualize JSON


```
{
    "id": 1,
    "book": {
        "id": 1,
        "title": "Effective Java",
        "author": "Joshua Bloch",
        "publicationYear": 2008,
        "isbn": "9780134685991"
    },
    "patron": {
        "id": 1,
        "name": "John Doe",
        "contactInformation": "john.doe@example.com"
    },
    "borrowDate": "2024-08-13",
    "returnDate": "2024-08-13"
}
```


7. Update Book Information

Endpoint:

- **Method:** `PUT /api/books/{id}`
- **URL:** `/api/books/1`
- **Request Body:{**

  "title": "Effective Java, 3rd Edition",

  "author": "Joshua Bloch",

  "publicationYear": 2018,

  "isbn": "9780134685991"

```
}
```

PUT http://localhost:8090/api/books/1

arams Authorization Headers (11) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "title": "Effective Java, 2nd Edition",
3   "author": "Joshua Bloch",
4   "publicationYear": 2018,
5   "isbn": "9780134685991"
6 }
7
```

dy Cookies (1) Headers (14) Test Results Status: 200 OK

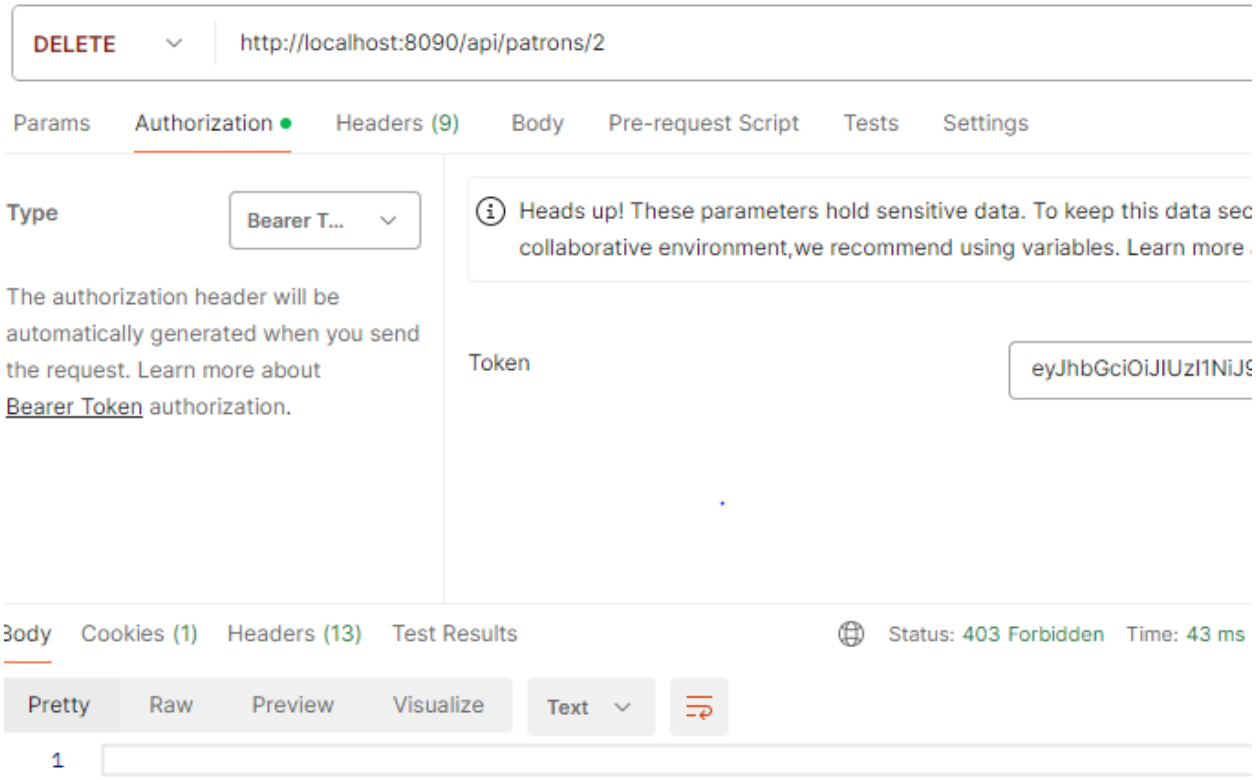
Pretty Raw Preview Visualize JSON

```
1 {
2     "id": 1,
3     "title": "Effective Java, 2nd Edition",
4     "author": "Joshua Bloch",
5     "publicationYear": 2018,
6     "isbn": "9780134685991"
7 }
```

8. Delete a Patron with id that doesnot exist in databse

Endpoint:

- **Method:** `DELETE /api/patrons/{id}`
- **URL:** `/api/patrons/2`
- **Expected Response:**

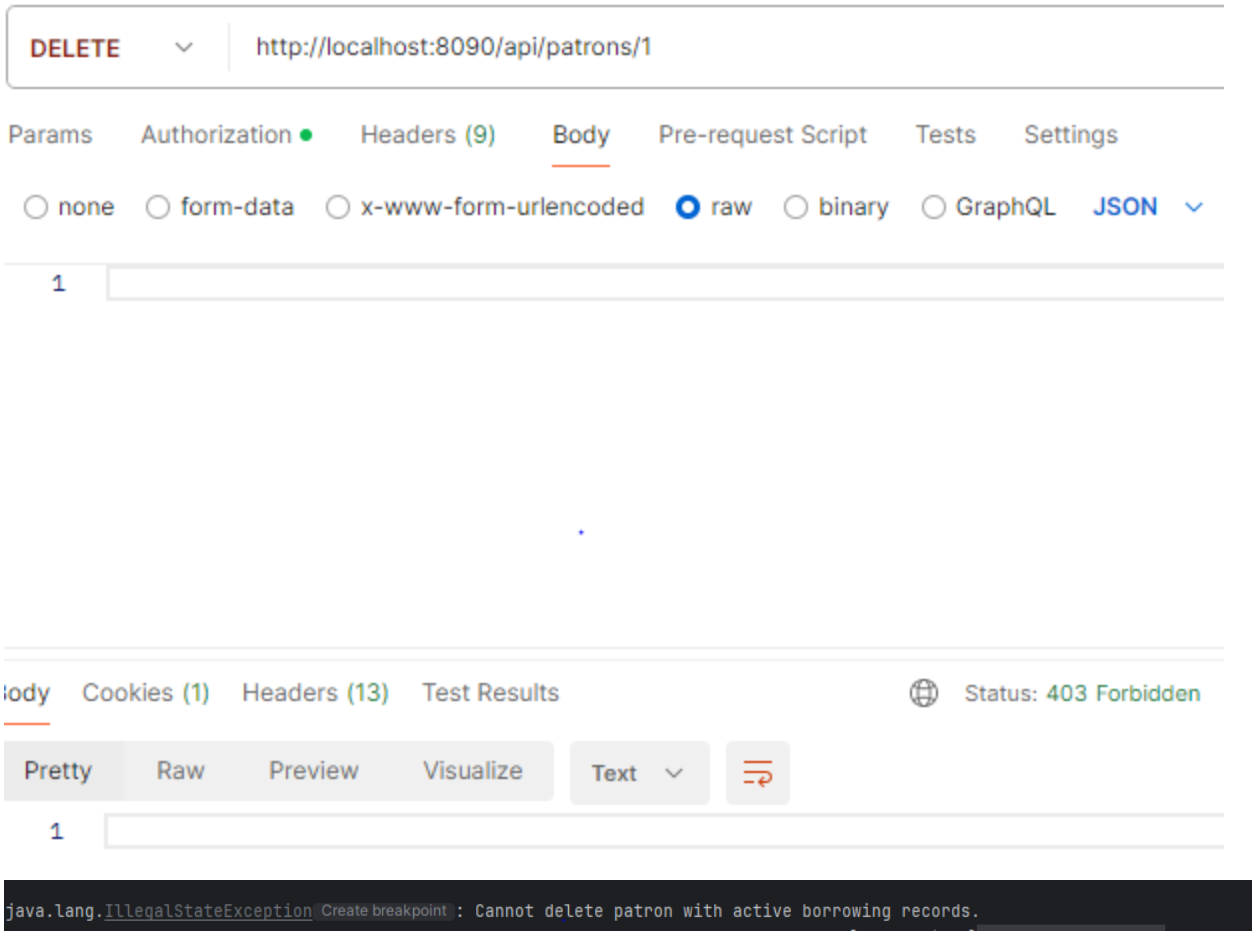



```
om.example.library.Borrowing.ResourceNotFoundException Create breakpoint : Patron not found with id: 2
```

8. Delete a Patron that did not yet return a book he/she borrowed

Endpoint:

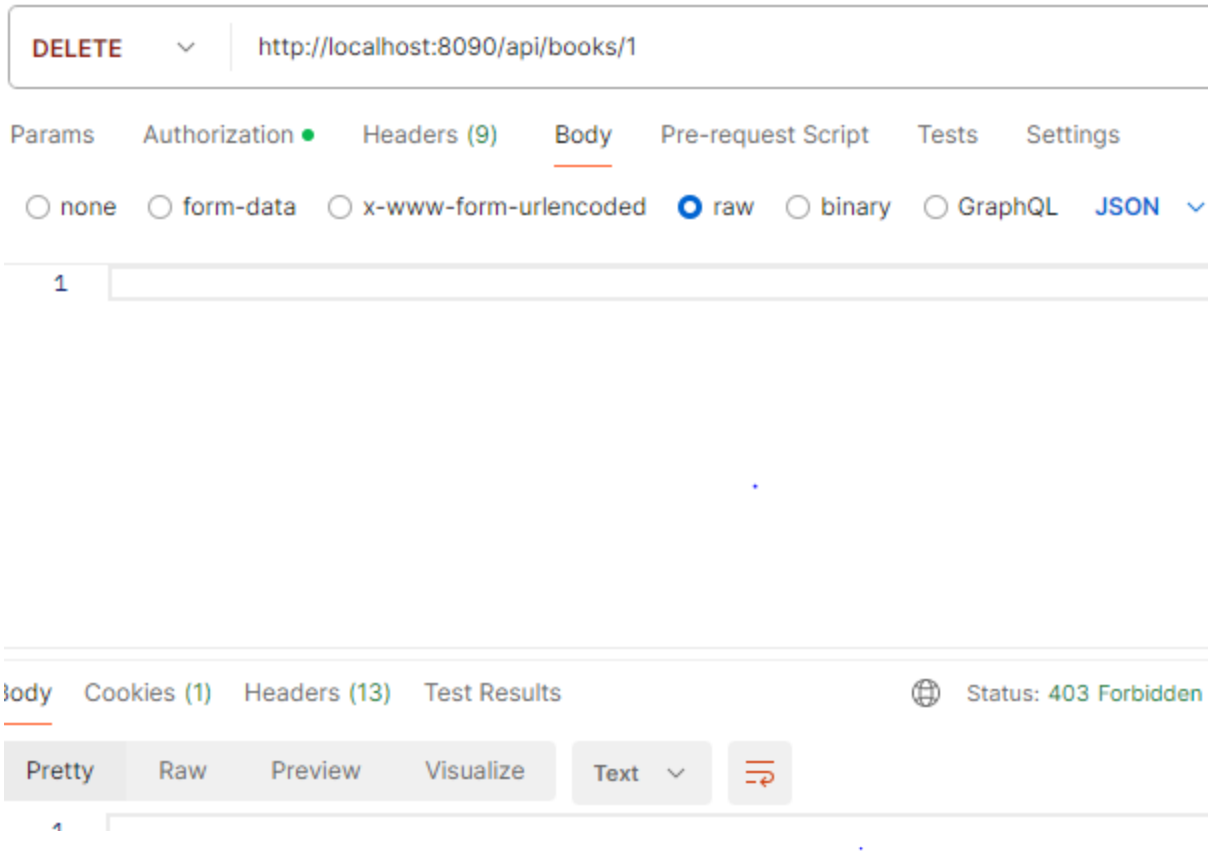
- **Method:** `DELETE /api/patrons/{id}`
- **URL:** `/api/patrons/1`
- **Expected Response:**



9. Delete a Book that is currently borrowed

Endpoint:

- **Method:** `DELETE /api/books/{id}`
- **URL:** `/api/books/1`
- **Expected Response:**




```
ava.lang.IllegalStateException Create breakpoint : Cannot delete book that is currently borrowed.
   at com.example.library.book.BookService.deleteBook(BookService.java:50) ~[classes/:na] <2 internal lines>
```

Unit Testing:
BookControllerTests.java
PatronControllerTests.java
BorrowingRecordControllerTests.java

Developed comprehensive unit tests for all API endpoints, including tests for authentication, authorization, and error handling scenarios (e.g., unauthorized access, invalid tokens). These tests ensure that security measures are correctly enforced across the application.

(config Package)
JWT Security Configuration:

JWT Creation: Detailed the process of token generation, including signing algorithms, token expiration, and payload structure.
Token Validation: Described how tokens are validated, including checking signatures, expiration, and claims.
Error Handling: Explained how security-related errors (e.g., invalid token, expired token) are handled and communicated to the client.
API Endpoint Protection:

Access Control: Specified which endpoints are protected by JWT and the roles or permissions required to access them.

Security Filters: Documented the security filters used (e.g., JwtAuthenticationFilter, JwtAuthorizationFilter) and their roles in the request processing pipeline.