Library Management System using Spring Boot

Project Overview

The Library Management System (LMS) is a Spring Boot application designed to manage books, patrons, and borrowing records. The application supports two types of users:

- **Patrons**: Can borrow books.
- **Librarians**: Can manage books and patrons (add, delete, update, get).

Both librarians and patrons can log in and register to access their respective functionalities. The application uses JWT (JSON Web Token) for authentication and role-based authorization to control access to different features.

Prerequisites

- **Java Development Kit (JDK)**: Ensure Java 17 SDK is installed on your machine.
- **MySQL Database**: Make sure MySQL is installed and running.
- **MySQL Workbench or Command-Line Interface**: For database management.

Setup Instructions

1. Create the MySQL Database

1. **Open MySQL Workbench or Command-Line Interface.**
2. **Create a new database named librarydb:**

   ```
   CREATE DATABASE librarydb;
   ```

2. Configure the Spring Boot Application

1. **Set Up Environment Variables**

You need to configure the environment variables for your MySQL database credentials. Create or modify the environment variables as follows:

   - MYSQL_USERNAME: Your MySQL username.
   - MYSQL_PASSWORD: Your MySQL password.

3. Build and Run the Application

Run the Project using IntelliJ ide or:

1. **Navigate to Your Project Directory**

Open a terminal or command prompt and navigate to the root directory of your Spring Boot project.

2. **Build the Project**

Use Maven to build the project:

```
mvn clean install
```

3. **Run the Application**

After building the project, run the application using the following command. For Maven:

```
mvn spring-boot:run
```

4. Access the Application

Once the application is running, you can access it through a web browser or API client. Typically, the application will be available at http://localhost:8080.

5. Testing and Usage

- **Register and Log In**: Both librarians and patrons can register and log in to access their respective functionalities.
- **Patrons**: Can browse and borrow books.
- **Librarians**: Can manage books and patrons (add, delete, get, update books).

Using the APIs Tutorial

The postman collection for this project is uploaded and can be found on github

How to Run Authentication APIs

The AuthController provides two main endpoints for user authentication: /register for user registration and /login for user login. Below is a detailed guide on how to interact with these APIs.

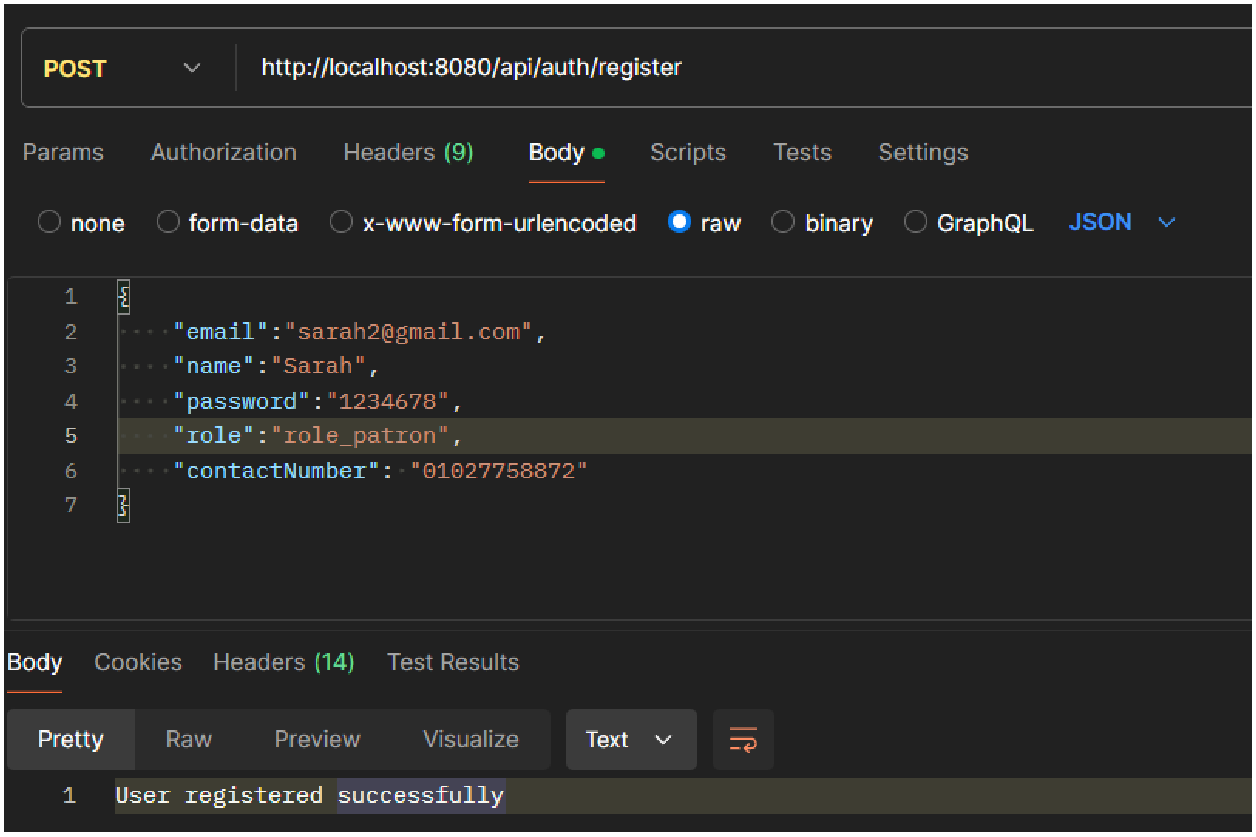
1. Register a New User

Endpoint: POST http://localhost:8080/api/auth/register

Description: This endpoint allows you to register a new user by providing the necessary details such as name, email, password, role, and contact information.

If you want to register as librarian, then write “role”=”role_librarian” in the json body.

If you want to register as librarian, then write “role”=”role_patron” in the json body



```
{
    "email":"sarah2@gmail.com",
    "name":"Sarah",
    "password":"1234678",
    "role":"role_patron",
    "contactNumber": "01027758872"
}
```

```
User registered successfully
```

Response:

- **Success:** Returns a 200 OK status with a success message.

```
{
  "message": "User registered successfully"
}
```

- **Failure:** Returns a 400 Bad Request status with an error message if any validation fails or if the user already exists.

```
{
  "message": "User already exists"
}
```

2. Login a User

Endpoint: POST /api/auth/login

Description: This endpoint allows a user to log in by providing a valid email and password. On success, it returns an authentication token and other relevant details.

Request Body:

```
{
  "email": "john.doe@example.com",
  "password": "securepassword"
}
```

Response:

- **Success:** Returns a 200 OK status with an AuthResponseDTO containing the authentication token and user details

```
{
    "accessToken": "eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJzYXJhaDJAZ21haWwuY29tIiwiaWF0IjoxNzIzMTk5MzIyLCJleHAiOjE3MjMyMDAwMjJ9.s_xcZq7vBKkwSI5jiT5831E5frwCtfrQBp0X5ocswdVkYALcIninwQ2yzGqlDUluNPosJWJTcrEi1CDQFxPdCw",
    "tokenType": "Bearer "
}
```

- **Failure:** Returns a 401 Unauthorized status if the credentials are invalid.

How to Use the Book Management APIs

The BookController provides a set of endpoints for managing books in the library system. Each API request must be authenticated using a Bearer token. Only users with the ROLE_LIBRARIAN can perform certain operations such as adding, updating, or deleting books.

```
  }

}
```

Use the token from the response in the Authorization header for subsequent API requests.

1. Get All Books

Endpoint: GET /api/books

Description: Fetches a list of all books in the library.

Authorization: Any authenticated user can access this endpoint.

Header:

Authorization: Bearer <your-access-token>

Response:

Success: Returns a 200 OK status with a list of books.

2. Get a Book by ID

Endpoint: GET /api/books/{id}

Description: Fetches a single book by its ID.

Authorization: Any authenticated user can access this endpoint.

Header:

Authorization: Bearer <your-access-token>

Response:

Success: Returns a 200 OK status with the book details.

Failure: Returns a 404 Not Found status if the book is not found.

3. Add a New Book

Endpoint: POST /api/books

Description: Adds a new book to the library. Only users with the ROLE_LIBRARIAN can perform this operation.

Authorization: Only users with the ROLE_LIBRARIAN.

Header:

bash

Copy code

```
Authorization: Bearer <your-access-token>

Content-Type: application/json
```

Request Body:

json

Copy code

```
{
  "title": "Effective Java",
  "author": "Joshua Bloch",
  "publicationYear": 2008,
  "isbn": "978-0134685991",
  "quantity": 10
}
```

Response:

- **Success:** Returns a 201 Created status with the created book details.

  json

  Copy code

  ```
  {
    "id": 1,
    "title": "Effective Java",
    "author": "Joshua Bloch",
    "publicationYear": 2008,
    "isbn": "978-0134685991"
  }
  ```

- **Failure:** Returns a 403 Forbidden status if the user does not have the ROLE_LIBRARIAN.

  ```
  { "message": "Access is denied"}
  ```

4. Update a Book

Endpoint: PUT /api/books/{id}

Description: Updates the details of an existing book. Only users with the ROLE_LIBRARIAN can perform this operation.

Authorization: Only users with the ROLE_LIBRARIAN.

Header:

Authorization: Bearer <your-access-token>

Content-Type: application/json

Request Body:

```
{
  "title": "Clean Code",
  "author": "Robert C. Martin",
  "publicationYear": 2008,
  "isbn": "978-0132350884",
  "quantity": 5
}
```

Response:

Success: Returns a 200 OK status with the updated book details.

Failure: Returns a 404 Not Found status if the book is not found, or a 403 Forbidden status if the user does not have the ROLE_LIBRARIAN.

}'

5. Delete a Book

Endpoint: DELETE /api/books/{id}

Description: Deletes a book from the library. Only users with the ROLE_LIBRARIAN can perform this operation.

Authorization: Only users with the ROLE_LIBRARIAN.

Header:

Authorization: Bearer <your-access-token>

Response:

Success: Returns a 204 No Content status if the book is deleted successfully.

Failure: Returns a 404 Not Found status if the book is not found, or a 403 Forbidden status if the user does not have the ROLE_LIBRARIAN.

Documentation on How to Use the Patron and Borrowing Management APIs

The PatronController and BorrowingController provide endpoints for managing patrons and book borrowings in the library system. Each API request requires authentication using a Bearer token and appropriate role-based access control.

1. Authentication

Pre-requisite: Before using any of the patron or borrowing management APIs, you must log in and obtain an access token.

- **Login Endpoint:** POST /api/auth/login
- **Authentication Type:** Bearer Token
- **Header:** Authorization: Bearer <your-access-token>

Use the token obtained from the login response in the Authorization header for subsequent API requests.

2. Patron Management

PatronController provides endpoints to manage patron details. Both ROLE_LIBRARIAN and ROLE_PATRON can access these endpoints.

Get All Patrons

Endpoint: GET /api/patrons

Description: Fetches a list of all patrons.

Authorization: Authenticated users with ROLE_LIBRARIAN or ROLE_PATRON can access this endpoint.

Header:

Authorization: Bearer <your-access-token>

Response:

- **Success:** Returns a 200 OK status with a list of patrons.
- **Failure:** Returns a 403 Forbidden status if the user does not have the required role.

Get a Patron by ID

Endpoint: GET /api/patrons/{id}

Description: Fetches details of a single patron by ID.

Authorization: Authenticated users with ROLE_LIBRARIAN or ROLE_PATRON can access this endpoint.

Header:

Authorization: Bearer <your-access-token>

Response:

- **Success:** Returns a 200 OK status with the patron details.
- **Failure:** Returns a 404 Not Found status if the patron is not found, or a 403 Forbidden status if the user does not have the required role.

Update a Patron

Endpoint: PUT /api/patrons/{id}

Description: Updates the details of an existing patron.

Authorization: Authenticated users with ROLE_LIBRARIAN or ROLE_PATRON can access this endpoint.

Header:

Authorization: Bearer <your-access-token>

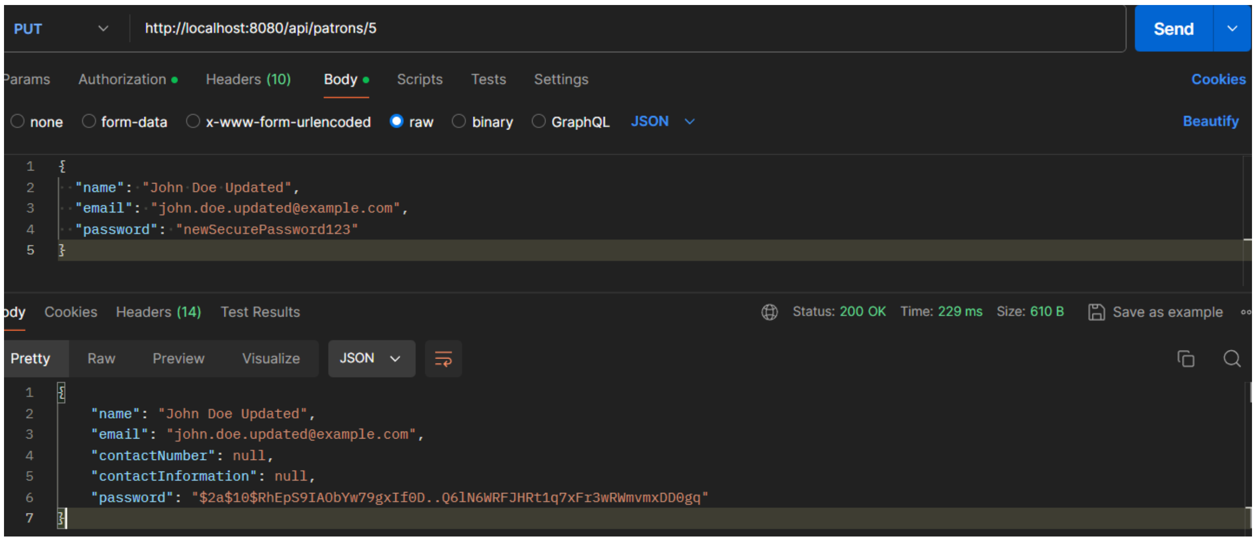
Content-Type: application/json

Request Body:

```
{
  "name": "Jane Doe",
  "email": "jane.doe@example.com",
  "contactNumber": "555-6789",
  "contactInformation": "123 Main St",
  "password": "newsecurepassword"
}
```

Response:

- **Success:** Returns a 200 OK status with the updated patron details.
- **Failure:** Returns a 404 Not Found status if the patron is not found, or a 403 Forbidden status if the user does not have the required role.



http://localhost:8080/api/patrons/5

```
{
  "name": "John Doe Updated",
  "email": "john.doe.updated@example.com",
  "password": "newSecurePassword123"
}
```

Status: 200 OK Time: 229 ms Size: 610 B

```
{
    "name": "John Doe Updated",
    "email": "john.doe.updated@example.com",
    "contactNumber": null,
    "contactInformation": null,
    "password": "$2a$10$RhEpS9IAObYw79gxIf0D..Q61N6WRFJHRt1q7xFr3wRWmvmxDD0gq"
}
```

Delete a Patron

Endpoint: DELETE /api/patrons/{id}

Description: Deletes a patron from the system.

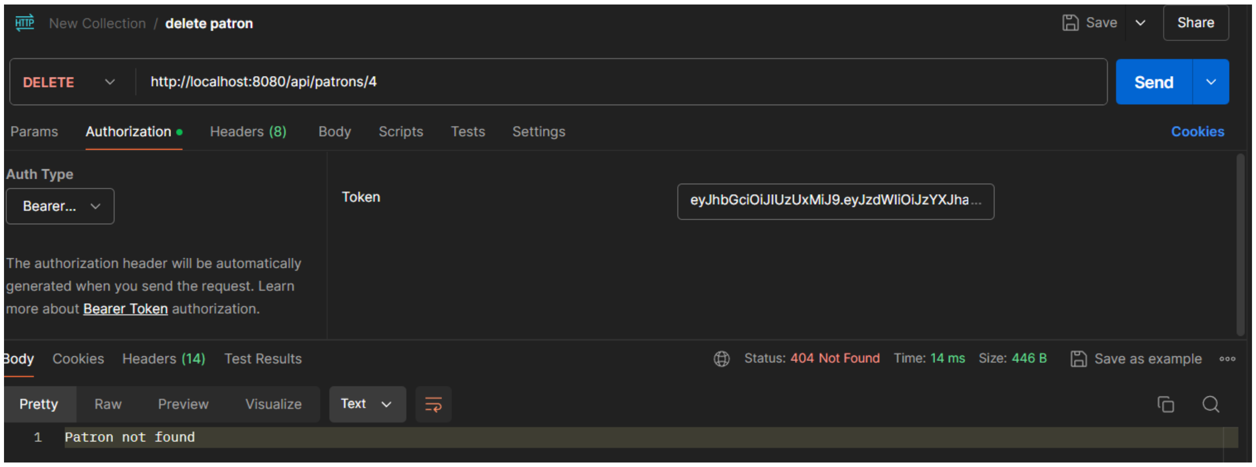
Authorization: Authenticated users with ROLE_LIBRARIAN or ROLE_PATRON can access this endpoint.

Header:

Authorization: Bearer <your-access-token>

Response:

- **Success:** Returns a 204 No Content status if the patron is deleted successfully.
- **Failure:** Returns a 404 Not Found status if the patron is not found, or a 403 Forbidden status if the user does not have the required role.



http://localhost:8080/api/patrons/4

Auth Type: Bearer...

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Token: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJzYXJha...

Status: 404 Not Found Time: 14 ms Size: 446 B

```
Patron not found
```

3. Borrowing Management

BorrowingController manages book borrowing operations. Only users with the ROLE_PATRON can perform these operations.

Borrow a Book

Endpoint: POST /api/borrow

Description: Allows a patron to borrow a book.

Authorization: Authenticated users with ROLE_PATRON.

Header:

Authorization: Bearer <your-access-token>

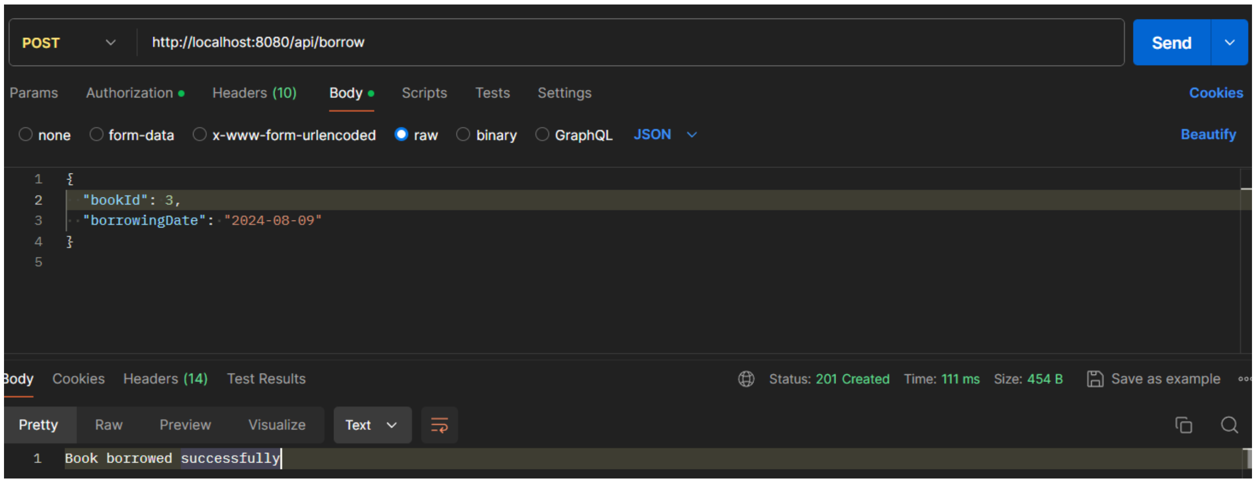
Content-Type: application/json

Request Body:

```
{
  "bookId": 1,
  "borrowingDate": "2024-08-09"
}
```

Response:

- **Success:** Returns a 201 Created status with a success message.
- **Failure:** Returns a 400 Bad Request status if the request is invalid, or a 403 Forbidden status if the user does not have the required role.

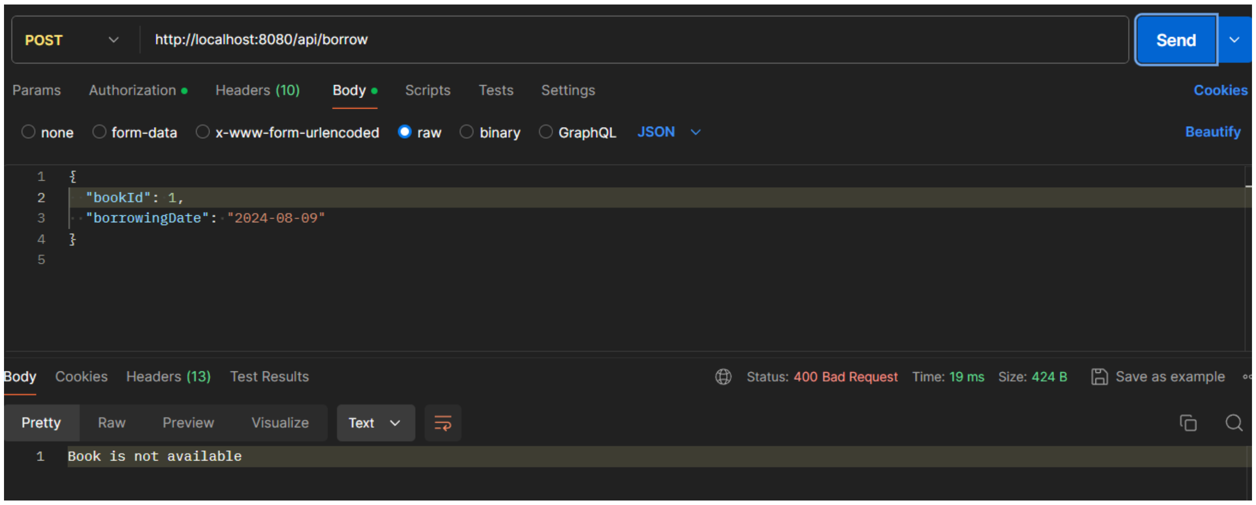


```
{
  "bookId": 3,
  "borrowingDate": "2024-08-09"
}
```

Status: 201 Created Time: 111 ms Size: 454 B

```
Book borrowed successfully
```

If book quantity is equal to zero:



http://localhost:8080/api/borrow

```
{
  "bookId": 1,
  "borrowingDate": "2024-08-09"
}
```

Status: 400 Bad Request Time: 19 ms Size: 424 B

```
Book is not available
```

Return a Book

Endpoint: PUT /api/borrow/return/{bookId}

Description: Allows a patron to return a borrowed book.

Authorization: Authenticated users with ROLE_PATRON.

Header:

Authorization: Bearer <your-access-token>

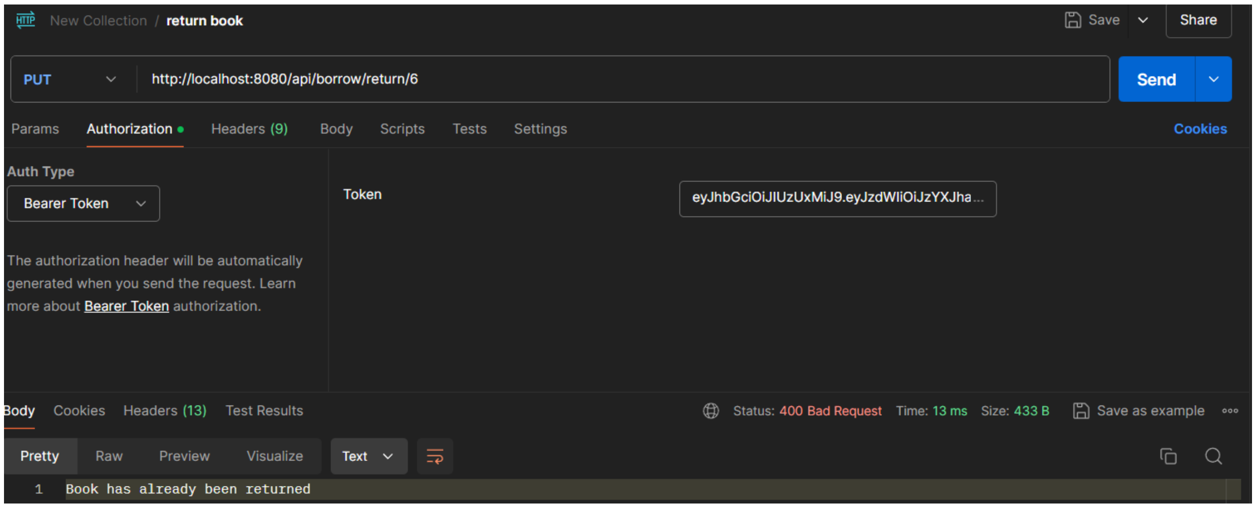
Response:

- **Success:** Returns a 200 OK status with a success message.
- **Failure:** Returns a 400 Bad Request status if the request is invalid, or a 403 Forbidden status if the user does not have the required role.

Example Response:

"Book returned successfully"

Error Handling:



PUT http://localhost:8080/api/borrow/return/6

Auth Type: Bearer Token

The authorization header will be automatically generated when you send the request. Learn more about Bearer Token authorization.

Token: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJzYXJha...

Status: 400 Bad Request Time: 13 ms Size: 433 B

```
Book has already been returned
```

Important Notes

- **Bearer Token:** Each request must include a valid Bearer token obtained after logging in.
- **Role-Based Access Control:**
  - ROLE_LIBRARIAN and ROLE_PATRON can manage patron details.
  - Only ROLE_PATRON can borrow and return books.
  - Only ROLE_LIBRARIAN can manage books (add, delete, edit).
- **Error Handling:** Ensure to handle responses such as 403 Forbidden, 404 Not Found, and 400 Bad Request appropriately in your application.