**Library Management System API Documentation**

**Project Overview**

Welcome to the Library Management System API! This API is designed to help librarians manage books, patrons, and borrowing records. With this RESTful API, you can perform all the necessary operations to keep your library running smoothly.

**Technologies Used**

* **Spring Boot**: The backbone of our API.
* **Spring Data JPA**: For seamless database interactions.
* **MySQL**: Our choice for data persistence.
* **Spring Security**: (Optional) For securing our API endpoints.
* **JUnit and Mockito**: Tools for unit testing.

**Getting Started**

**Prerequisites**

Before you dive in, make sure you have the following installed:

* Java Development Kit (JDK) 8 or higher.
* Maven .
* A running MySQL server.

**Setup Instructions**

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

bash

Copy code

git clone https://github.com/yourusername/library-management-system.git

cd library-management-system

1. **Create the MySQL Database:** Open your MySQL client and create the database:

sql

Copy code

CREATE DATABASE maidlibrarydb;

1. **Configure Database Connection:** Next, you’ll need to set up your database connection. Open src/main/resources/application.properties and add your MySQL credentials:

properties

Copy code

1. spring.datasource.url=jdbc:mysql://localhost:3306/maidlibrarydb
2. spring.datasource.username=root
3. #spring.datasource.password=M@id123cc
4. spring.jpa.hibernate.ddl-auto=update
5. spring.jpa.show-sql=true
6. spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
7. spring.jackson.serialization.fail-on-empty-beans=false
8. **Build the Project:** If you’re using Maven, you can build the project with:

bash

Copy code

mvn clean install

1. **Run the Application:** Now, start the application:

bash

Copy code

mvn spring-boot:run

**API Endpoints**

**Base URL**

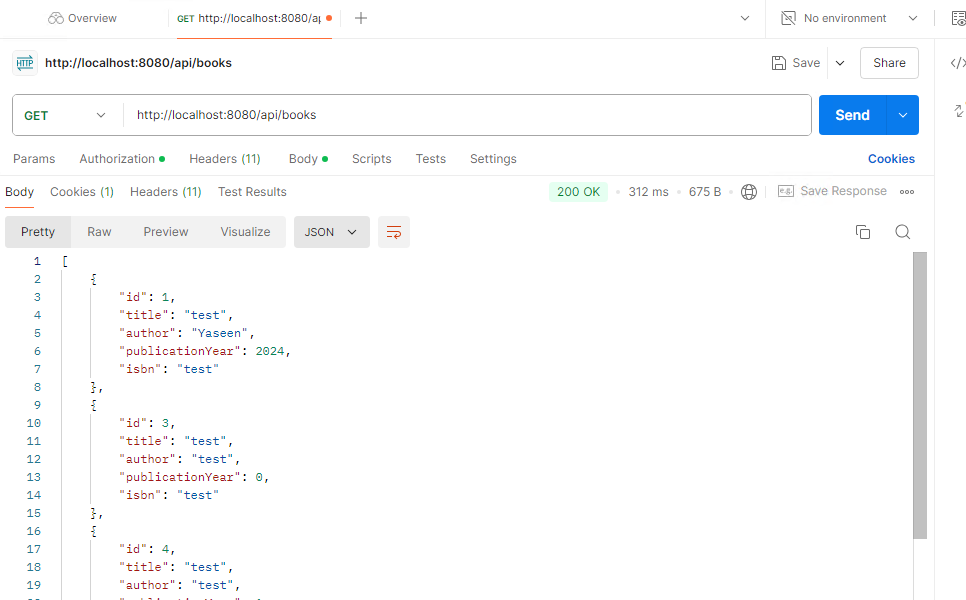
All the endpoints are available at:bash

Copy code

http://localhost:8080/api

**Book Management Endpoints**

* **Get All Books**

****

* + **URL**: /books
  + **Method**: GET
  + **Response**: A list of all books in the library.
* **Get Book by ID**
  + **URL**: /books/{id}
  + **Method**: GET
  + **Response**: Details of the book specified by the ID.
* **Add New Book**
  + **URL**: /books
  + **Method**: POST
  + **Request Body**:

json

Copy code

{

"title": "Book Title",

"author": "Author Name",

"publicationYear": 2023,

"isbn": "123-4567890123"

}

* + **Response**: The details of the created book.
* **Update Book**
  + **URL**: /books/{id}
  + **Method**: PUT
  + **Request Body**: Similar to adding a new book.
  + **Response**: The updated book details.
* **Delete Book**
  + **URL**: /books/{id}
  + **Method**: DELETE
  + **Response**: Status 204 No Content if successful.

**Patron Management Endpoints**

* **Get All Patrons**
  + **URL**: /patrons
  + **Method**: GET
  + **Response**: A list of all patrons.
* **Get Patron by ID**
  + **URL**: /patrons/{id}
  + **Method**: GET
  + **Response**: Details of the patron specified by the ID.
* **Add New Patron**
  + **URL**: /patrons
  + **Method**: POST
  + **Request Body**:

json

Copy code

{

"name": "Patron Name",

"contactInfo": "patron@example.com"

}

* + **Response**: The details of the created patron.
* **Update Patron**
  + **URL**: /patrons/{id}
  + **Method**: PUT
  + **Request Body**: Similar to adding a new patron.
  + **Response**: The updated patron details.
* **Delete Patron**
  + **URL**: /patrons/{id}
  + **Method**: DELETE
  + **Response**: Status 204 No Content if successful.

**Borrowing Management Endpoints**

* **Borrow a Book**
  + **URL**: /borrow/{bookId}/patron/{patronId}
  + **Method**: POST
  + **Response**: Status 201 Created if successful.
* **Return a Book**
  + **URL**: /return/{bookId}/patron/{patronId}
  + **Method**: PUT
  + **Response**: Status 200 OK if successful.

**Error Handling**

We aim to handle errors gracefully, returning appropriate HTTP status codes:

* **400 Bad Request**: When input validation fails (e.g., missing required fields).
* **401 Unauthorized**: When authentication fails.
* **403 Forbidden**: When access is denied.
* **404 Not Found**: When a resource (like a book or patron) doesn’t exist.
* **500 Internal Server Error**: For unexpected server errors.

**Testing**

To ensure everything works as intended, you can write unit tests using JUnit and Mockito. Here’s how you can run your tests:

bash

Copy code

mvn test

**Conclusion**

That’s it! This API provides a solid foundation for managing a library. You can always expand it with more features, such as user authentication or caching for improved performance.

**Further Enhancements (Optional)**

* Consider implementing user authentication (either basic or JWT).
* Add caching for frequently accessed data.
* Introduce logging and monitoring for API calls to track usage and performance.

This project has been successfully completed as part of the quiz for the Back-End Developer position at maids.cc. I appreciate the opportunity to showcase my skills. Please excuse any incomplete aspects or weaknesses in the project due to the limited time available. I look forward to your feedback.

Yaseen Saad Aldeen