

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**

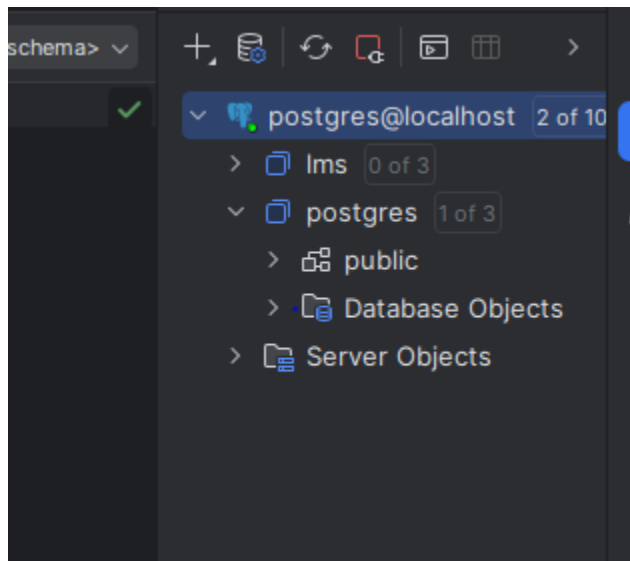
Running the Application

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

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

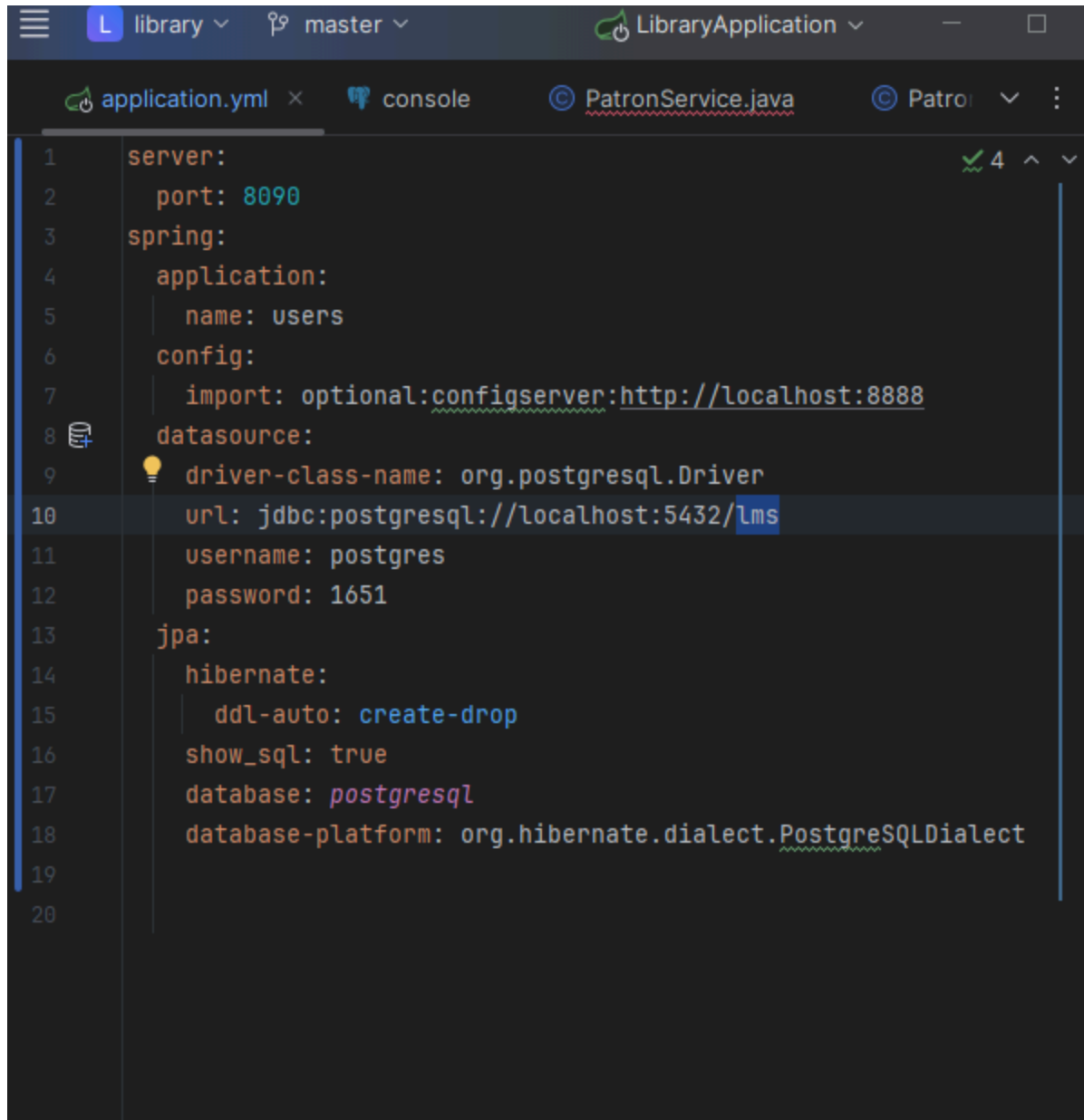
2. **Open in IntelliJ IDEA Ultimate:** Import the project into IntelliJ IDEA Ultimate.
3. **Configure PostgreSQL:** Set up PostgreSQL on your local machine and update the database configuration in the `application.yml` file.
 - + -> Datasource -> postgresQL

Create ->new ->database -> write database name "lms"



2 of 10 -> check lms

0 of 3 -> check all schemas under lms database



```
1  server:
2    port: 8090
3  spring:
4    application:
5      name: users
6    config:
7      import: optional:configserver:http://localhost:8888
8  datasource:
9    driver-class-name: org.postgresql.Driver
10   url: jdbc:postgresql://localhost:5432/lms
11   username: postgres
12   password: 1651
13  jpa:
14    hibernate:
15      ddl-auto: create-drop
16    show_sql: true
17    database: postgresql
18    database-platform: org.hibernate.dialect.PostgreSQLDialect
19
20
```

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.

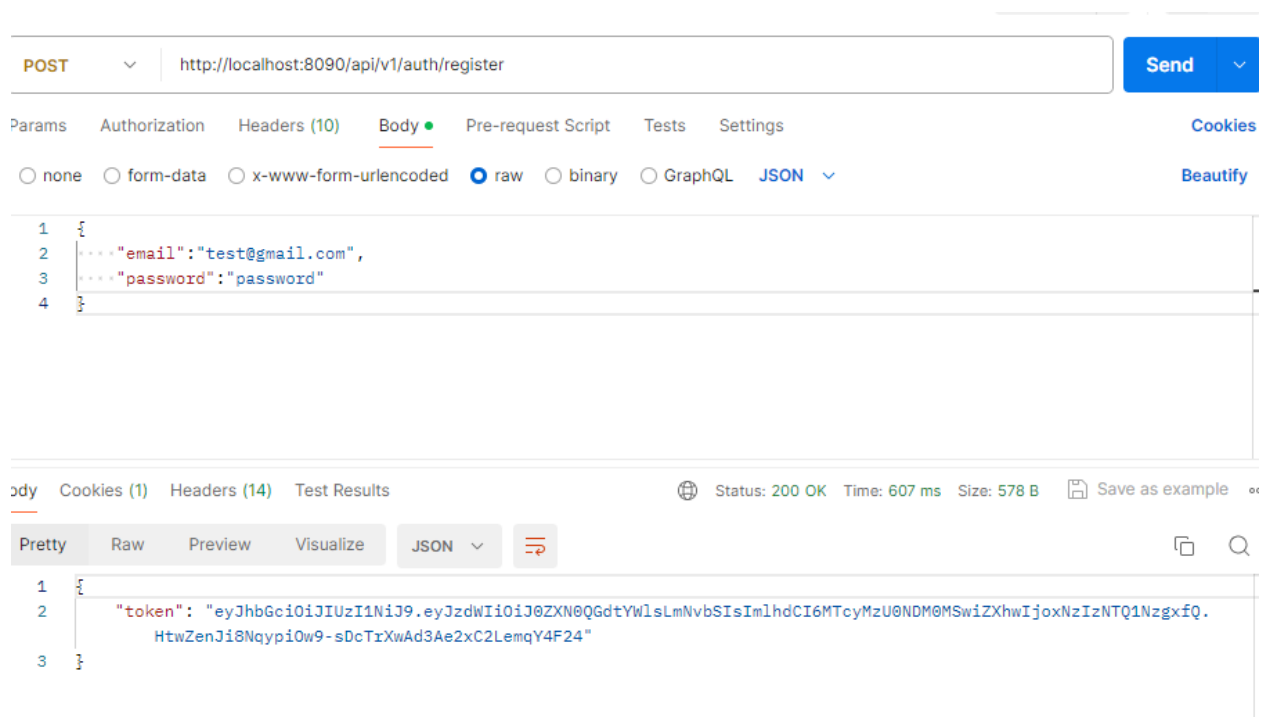
Interacting with API Endpoints using postman:

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

ParamsAuthorization ●Headers (11)Body ●Pre-request ScriptTestsSettings

Type

Bearer T... ▼

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

ⓘ Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about [variables](#).

Token

eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0QGdtYWlsLmNvbSIsImIhdCI6MTcyMzU0NTE1MCwiZXhwIjoxNzIzNTQ2NTkwfQ.hGi1dazKpX2kuXxfLcuyLkbPjh2FvRdf9-2WGHrV2Ik

POST

http://localhost:8090/api/books

ParamsAuthorizationHeaders (11)BodyPre-request ScriptTestsSettings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

{

2

"title": "Effective Java",

3

"author": "Joshua Bloch",

4

"publicationYear": 2008,

5

"isbn": "9780134685991"

6

}

7

bodyCookies (1)Headers (14)Test Results

Status: 200 OKTime: 156 msSize: 526 B

PrettyRawPreviewVisualizeJSON

1

{

2

"id": 1,

3

"title": "Effective Java",

4

"author": "Joshua Bloch",

5

"publicationYear": 2008,

6

"isbn": "9780134685991"

7

}

Activate

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"

}
```

POST

▼

http://localhost:8090/api/patrons

ParamsAuthorization ●Headers (11)Body ●Pre-request ScriptTestsSettings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

1

{

2

"name": "John Doe",

3

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

4

}

5

bodyCookies (1)Headers (14)Test Results

⌐Status: 200 OK

PrettyRawPreviewVisualizeJSON ▼⌵

1

{

2

"id": 1,

3

"name": "John Doe",

4

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

3. Retrieve All Books

Endpoint:

- Method: GET /api/books

GET

▼

http://localhost:8090/api/books

ParamsAuthorization ●Headers (11)Body ●Pre-request ScriptTestsSettings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON ▼

1

{

2

"name": "John Doe",

3

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

4

}

5

odyCookies (1)Headers (14)Test Results

⌚ Status: 200 OKTime: 47 ms

PrettyRawPreviewVisualizeJSON ▼

≡

1

[

2

{

3

"id": 1,

4

"title": "Effective Java",

5

"author": "Joshua Bloch",

6

"publicationYear": 2008,

7

"isbn": "9780134685991"

8

}

9

]

4. Retrieve All Patrons

Endpoint:

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

GET

▼

http://localhost:8090/api/patrons

ParamsAuthorization ●Headers (9)BodyPre-request ScriptTestsSettings

☐ none☐ form-data☐ x-www-form-urlencoded☒ raw☐ binary☐ GraphQLJSON ▼

1

odyCookies (1)Headers (14)Test Results

⌚ Status: 200 OKTime: 25 ms


PrettyRawPreviewVisualizeJSON ▼


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

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

Pretty Raw Preview Visualize JSON  

```
1 {
2   "id": 1,
3   "book": {
4     "id": 1,
5     "title": "Effective Java",
6     "author": "Joshua Bloch",
7     "publicationYear": 2008,
8     "isbn": "9780134685991"
9   },
10  "patron": {
11    "id": 1,
12    "name": "John Doe",
13    "contactInformation": "john.doe@example.com"
14  },
15  "borrowDate": "2024-08-13",
16  "returnDate": null
17 }
```

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

ParamsAuthorization ●Headers (10)BodyPre-request ScriptTestsSettings

bodyCookies (1)Headers (14)Test Results

🌐 Status: 200 OK

PrettyRawPreviewVisualizeJSON ▼

≡

1

{

2

"id": 1,

3

"book": {

4

"id": 1,

5

"title": "Effective Java",

6

"author": "Joshua Bloch",

7

"publicationYear": 2008,

8

"isbn": "9780134685991"

9

},

10

"patron": {

11

"id": 1,

12

"name": "John Doe",

13

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

14

},

15

"borrowDate": "2024-08-13",

16

"returnDate": "2024-08-13"

17

}

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"}
```

}

The screenshot displays a REST client interface. At the top, a PUT request is configured for the URL `http://localhost:8090/api/books/1`. The 'Body' tab is selected, showing a JSON payload for a book. Below the request, the 'Test Results' tab is active, showing a successful 200 OK response with a JSON object containing the book's details and its ID.

Request:

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

Response:

```
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:**

DELETE ▼ | http://localhost:8090/api/patrons/2

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

Type: Bearer T... ▼

The authorization header will be automatically generated when you send the request. Learn more about [Bearer Token](#) authorization.

Token: eyJhbGciOiJIUzI1NiJ9...

Heads up! These parameters hold sensitive data. To keep this data secure in our collaborative environment, we recommend using variables. [Learn more](#)

Body | **Cookies (1)** | Headers (13) | Test Results

Status: 403 Forbidden | Time: 43 ms

Pretty | Raw | Preview | Visualize | Text ▼ |


1


```
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:**



DELETE  http://localhost:8090/api/patrons/1


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

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

1 

body Cookies (1) Headers (13) Test Results  Status: 403 Forbidden

Pretty Raw Preview Visualize Text  

1 

```
java.lang.IllegalStateException Create breakpoint : Cannot delete patron with active borrowing records.
```

9. Delete a Book that is currently borrowed

Endpoint:

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

DELETE

▼

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

ParamsAuthorization●Headers (9)BodyPre-request ScriptTestsSettings

☐ none☐ form-data☐ x-www-form-urlencoded☒ raw☐ binary☐ GraphQLJSON▼

1

bodyCookies (1)Headers (13)Test Results

⌚ Status: 403 Forbidden

PrettyRawPreviewVisualizeText▼

⌕

```
java.lang.IllegalStateException: 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.