

Library Management System

Overview:

The **Library Management System** is a Spring Boot application designed to manage the operations of a library, including book management, patron management, and borrowing records. This system provides RESTful APIs for interaction with the backend services.

Prerequisites:

- Java Development Kit (JDK) 8 or higher
- Maven 3.6 or higher
- MySQL database
- Git

Installation

1. Clone the repository:

```
git clone https://github.com/EslamA99/Library-Management-System.git cd Library-Management-System
```

2. Update the application.properties file located in src/main/resources to match your MySQL database configuration:

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/library_management
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.hibernate.ddl-auto=update
```

3. Build the project:

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

4. Run the application using the following command:

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

The application should now be running on <http://localhost:8080>.

API Endpoints:

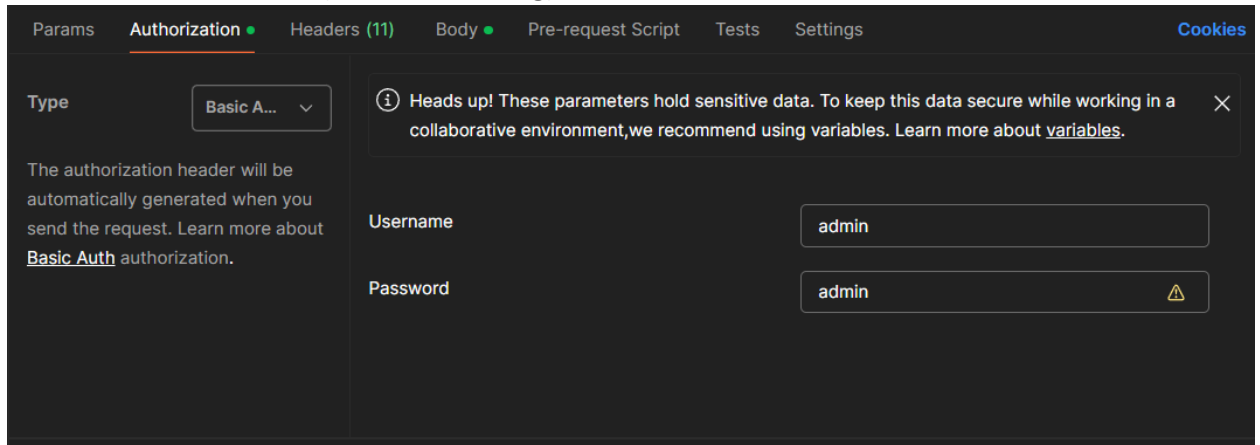
Before try the APIs please use this to users to authenticate to run all APIs.

Basic Auth section in postman:-

Username: admin

Password:admin

Admin can access all APIs (books, Borrowing)



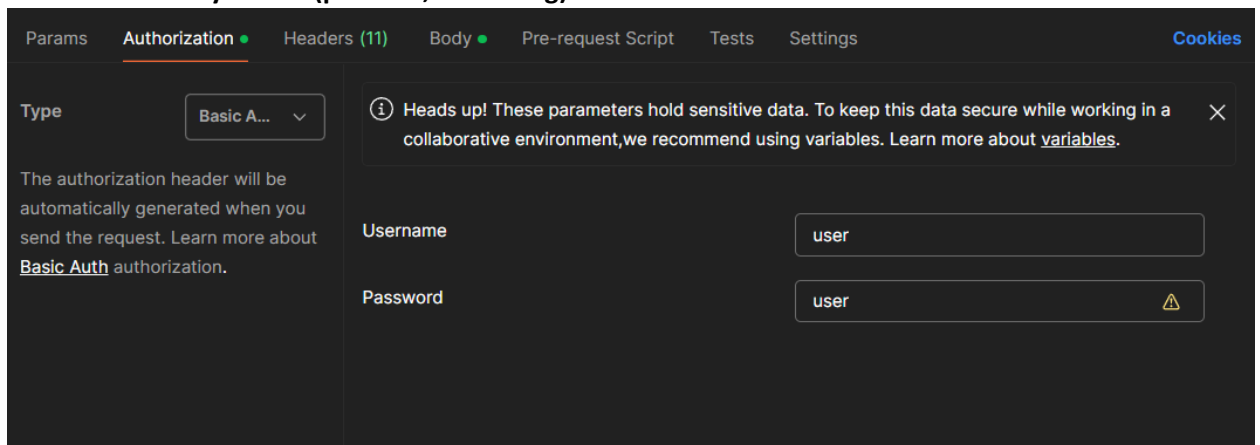
The image shows the Postman interface with the 'Authorization' tab selected. The 'Type' is set to 'Basic Auth'. A warning message states: 'Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. Learn more about [variables](#).' The 'Username' field contains 'admin' and the 'Password' field contains 'admin' with a warning icon.

Params	Authorization	Headers (11)	Body	Pre-request Script	Tests	Settings	Cookies
<p>Type: Basic A... ▾</p> <p>The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.</p> <p>Username: admin</p> <p>Password: admin ⚠</p>							

Username:user

Password:user

This user can only access (patrons ,Borrowing)



The image shows the Postman interface with the 'Authorization' tab selected. The 'Type' is set to 'Basic Auth'. A warning message states: '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](#).' The 'Username' field contains 'user' and the 'Password' field contains 'user' with a warning icon.

Params	Authorization	Headers (11)	Body	Pre-request Script	Tests	Settings	Cookies
<p>Type: Basic A... ▾</p> <p>The authorization header will be automatically generated when you send the request. Learn more about Basic Auth authorization.</p> <p>Username: user</p> <p>Password: user ⚠</p>							

Book Management Endpoints:

- **GET /api/books:** Retrieve a list of all books.

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

- URL:** `localhost:8080/api/books`
- Method:** `GET`
- Body:** `none` (selected)
- Status:** `200 OK`
- Time:** `3.01 s`
- Size:** `324 B`
- Response Body (JSON):**

```
[
  {
    "id": 1,
    "title": "book1",
    "author": "eslam",
    "publicationYear": 2020,
    "isbn": "1111"
  },
  {
    "id": 2,
    "title": "book2",
    "author": "eslam1",
    "publicationYear": 2022,
    "isbn": "2222"
  }
]
```

- **GET /api/books/{id}**: Retrieve details of a specific book by ID.

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

- Header:** libManagement / **getBookById**
- Method and URL:** GET localhost:8080/api/books/1
- Buttons:** Save, Send, and a dropdown menu.
- Tabs:** Params, Authorization, Headers (7), Body, Pre-request Script, Tests, Settings, Cookies.
- Query Params Table:**

Key	Value	Description	...	Bulk Edit
Key	Value	Description		
- Body Tab:** Cookies, Headers (5), Test Results. Status: 200 OK, 9 ms, 242 B. Action: Save as example.
- Body Content (JSON):**

```
1 {
2   "id": 1,
3   "title": "book1",
4   "author": "eslam",
5   "publicationYear": 2020,
6   "isbn": "1111"
7 }
```

- **POST /api/books:** Add a new book to the library.

The screenshot displays a REST client interface for a project named 'libManagement' with the endpoint 'addBook'. The request is a POST to 'localhost:8080/api/books'. The body is set to 'raw' and contains the following JSON:

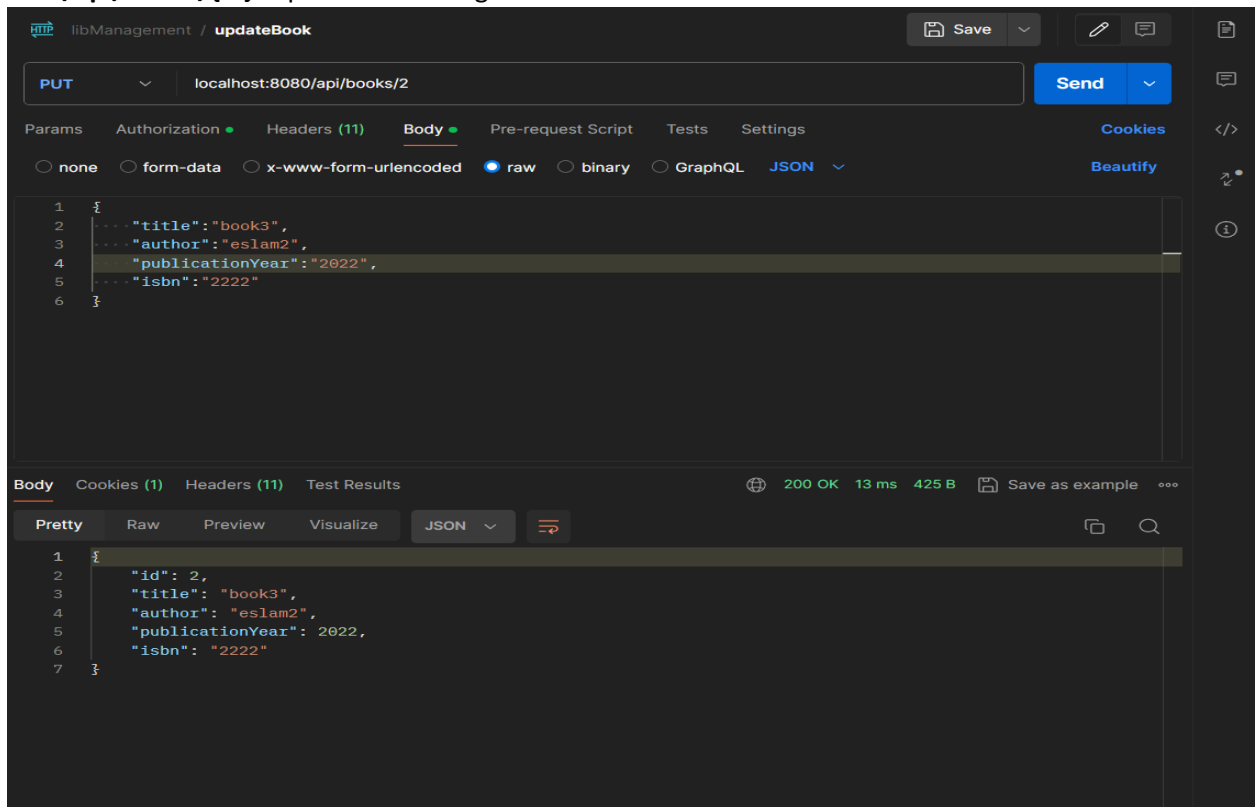
```
1 {
2   ... "title": "book3",
3   ... "author": "eslam2",
4   ... "publicationYear": "2020",
5   ... "isbn": "2222"
6 }
```

The response section shows the following JSON:

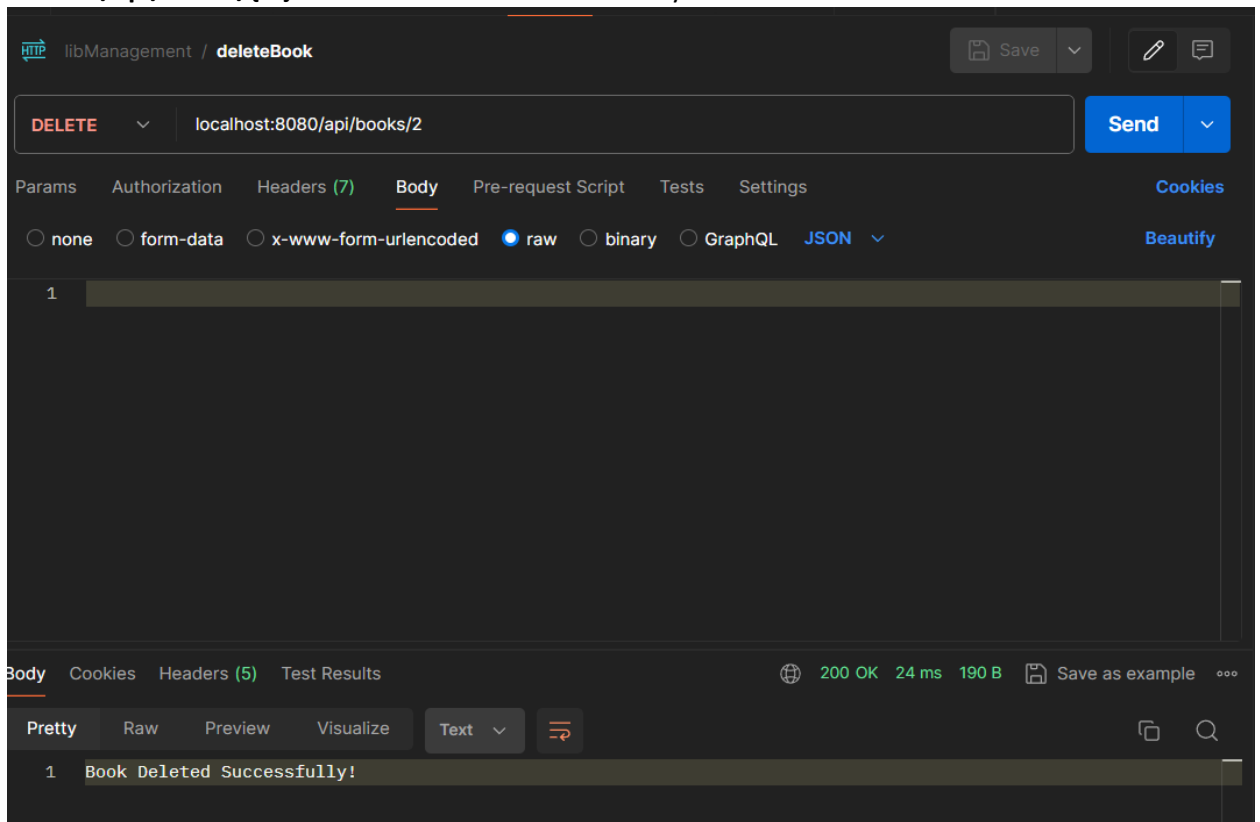
```
1 {
2   "id": 2,
3   "title": "book3",
4   "author": "eslam2",
5   "publicationYear": 2020,
6   "isbn": "2222"
7 }
```

Response details: 201 Created, 122 ms, 430 B. The interface includes tabs for Params, Authorization, Headers (11), Body, Pre-request Script, Tests, and Settings. The Body tab is active, showing the request and response JSON. The response is formatted as 'Pretty' JSON.

- **PUT /api/books/{id}**: Update an existing book's information.



- **DELETE /api/books/{id}**: Remove a book from the library.



Patron Management Endpoints:

- **GET /api/patrons:** Retrieve a list of all patrons.

The screenshot shows the Postman interface for a GET request to `localhost:8080/api/patrons`. The request is configured with Basic Authentication, where both the Username and Password are set to `user`. A warning message states: "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." The response status is `200 OK` with a response time of `97 ms` and a body size of `399 B`. The response body is displayed in JSON format:

```
[
  {
    "id": 1,
    "name": "eslam",
    "contactInfo": "01114213042"
  }
]
```

- **GET /api/patrons/{id}:** Retrieve details of a specific patron by ID.

The screenshot shows the Postman interface for a GET request to `localhost:8080/api/patrons/1`. The request is configured with Basic Authentication, where both the Username and Password are set to `user`. A warning message states: "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." The response status is `200 OK` with a response time of `12 ms` and a body size of `397 B`. The response body is displayed in JSON format:

```
{
  "id": 1,
  "name": "eslam",
  "contactInfo": "01114213042"
}
```

- **POST /api/patrons:** Add a new patron to the system.

The screenshot shows a REST client interface for a POST request to `localhost:8080/api/patrons`. The request body is a JSON object: `{ "name": "adel", "contactInfo": "01204213042" }`. The response status is `201 Created` with a response time of `21 ms` and a body size of `401 B`. The response body is a JSON object: `{ "id": 3, "name": "adel", "contactInfo": "01204213042" }`.

```
1 {
2   "name": "adel",
3   "contactInfo": "01204213042"
4 }
```

Body: `{ "id": 3, "name": "adel", "contactInfo": "01204213042" }`

- **PUT /api/patrons/{id}:** Update an existing patron's information.

The screenshot shows a REST client interface for a PUT request to `localhost:8080/api/patrons/3`. The request body is a JSON object: `{ "name": "ahmed", "contactInfo": "0111556677" }`. The response status is `200 OK` with a response time of `13 ms` and a body size of `396 B`. The response body is a JSON object: `{ "id": 3, "name": "ahmed", "contactInfo": "0111556677" }`.

```
1 {
2   "name": "ahmed",
3   "contactInfo": "0111556677"
4 }
```

Body: `{ "id": 3, "name": "ahmed", "contactInfo": "0111556677" }`

- **DELETE /api/patrons/{id}**: Remove a patron from the system.

The screenshot shows the Postman interface for a DELETE request. The URL is `localhost:8080/api/patrons/3`. The response status is **200 OK** with a response time of **25 ms** and a body size of **374 B**. The response body, viewed in 'Text' format, contains the message: `1 patron Deleted Successfully!`.

Borrowing Endpoints:

- **POST /api/borrow/{bookId}/patron/{patronId}**: Allow a patron to borrow a book.

The screenshot shows the Postman interface for a POST request. The URL is `localhost:8080/api/borrow/2/patron/1`. The 'Authorization' tab is active, showing 'Basic Auth' with the username `admin` and password `admin`. The response status is **201 Created** with a response time of **37 ms** and a body size of **449 B**. The response body, viewed in 'JSON' format, is:

```

1 {
2   "id": 4,
3   "borrowingDate": "2024-08-08T01:02:11.016+00:00",
4   "returnDate": null,
5   "bookId": 2,
6   "patronId": 1
7 }

```

- **PUT /api/return/{bookId}/patron/{patronId}**: Record the return of a borrowed book by a patron.

libManagement / Borrow record / return

PUT localhost:8080/api/return/2/patron/1

Send

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

Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies (1) Headers (11) Test Results

200 OK 17 ms 471 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 4,
3   "borrowingDate": "2024-08-08T01:02:11.016+00:00",
4   "returnDate": "2024-08-08T01:03:43.004+00:00",
5   "bookId": 2,
6   "patronId": 1
7 }
```

Database Schema

The database schema includes tables for books, patrons, and borrowing records. Here is a basic schema description:

- **Books:**
 - id (Primary Key)
 - title
 - author
 - publicationYear
 - isbn
- **Patrons:**
 - id (Primary Key)
 - name
 - contactInformation
- **Borrowing Records:**
 - id (Primary Key)
 - book_id (Foreign Key to Books)
 - patron_id (Foreign Key to Patrons)
 - borrowDate
 - returnDate