Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

LIBRARY MANAGEMENT SYSTEM

Overview:

This document provides instructions on how to run the Library Management application, interact with its API endpoints through Postman, and utilize authentication mechanisms.

Getting started

Prerequisites

- Java JDK 17 or above
- MYSQL Server
- Postman for testing API endpoints

Application Setup:

1. Clone the repository to your local machine.
2. Ensure MySQL is running and create a database named library_management.
    - Connect Spring to MYSQL using driver spring.datasource.url=jdbc:mysql://localhost:3306/library_management
3. Update application.properties with your MySQL username and password:
    - spring.datasource.username=#username
    - spring.datasource.password=#password

   Navigate to the project's root directory in your terminal or command prompt.
4. Run the application using:

```
./mvnw spring-boot:run
      or
 mvnw spring-boot:run
```

Interacting with the API using Postman

Importing the Collection

1. Launch Postman.
2. Click on Import button.
3. Choose Link and paste the link to your API collection if you have it hosted. Alternatively, you can manually create a new collection within Postman.

Authentication

This application uses Basic Authentication for protected endpoints.All endpoints are secured and below details are needed for Authorization.

- **Username: admin**
- **Password: secret**

To Configure these username and password details, change them in application.properties file

- spring.security.user.name=#username
- spring.security.user.password=#password

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

To authenticate in **Postman**:

1. Select the request.
2. Go to the Authorization tab.
3. Choose Basic Auth from the Type dropdown.
4. Enter the username and password

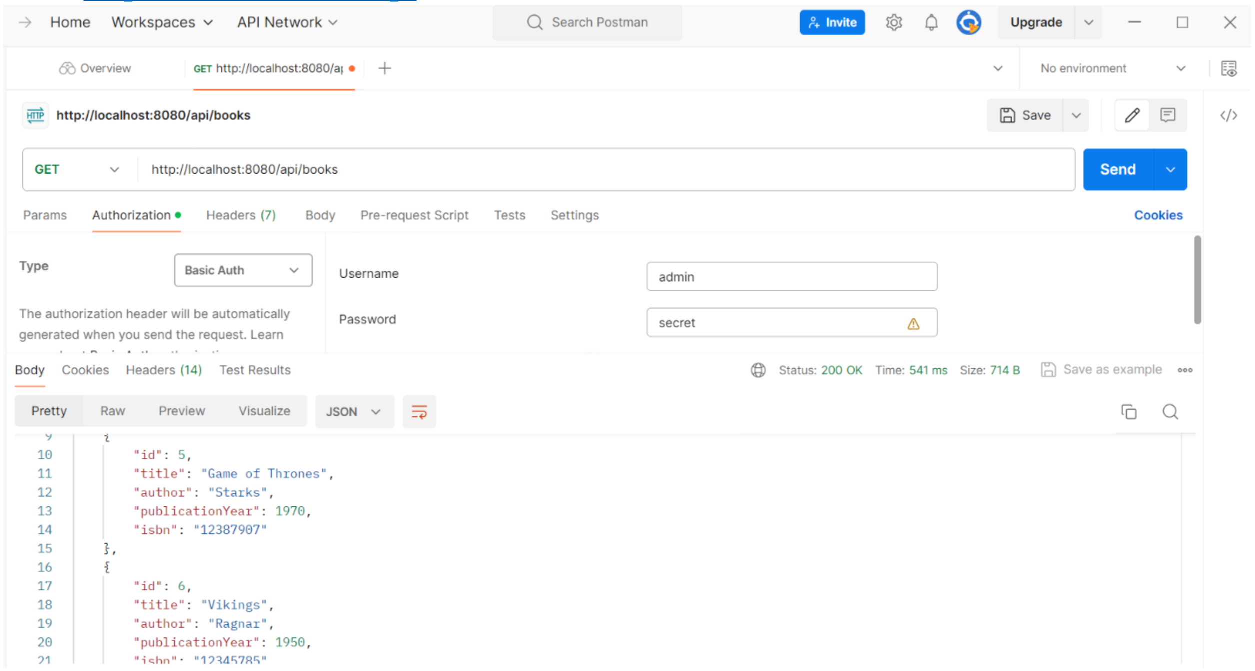
Making Requests

All requests are made to the URL: http://localhost:8080/api

Public Endpoints for Book Entity

1)Get All Books

Retrieves a list of all books.
Method: GET
URL: http://localhost:8080/api/books



http://localhost:8080/api/books

GET http://localhost:8080/api/books

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn

Username: admin
Password: secret

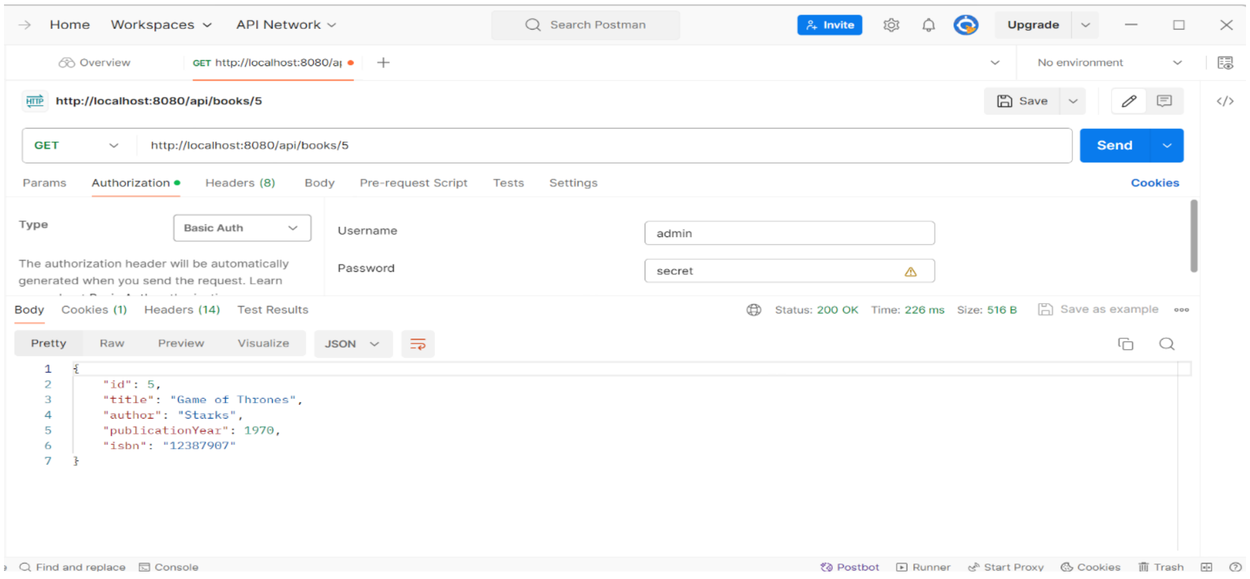
Status: 200 OK Time: 541 ms Size: 714 B

```
    {
        "id": 5,
        "title": "Game of Thrones",
        "author": "Starks",
        "publicationYear": 1970,
        "isbn": "12387907"
    },
    {
        "id": 6,
        "title": "Vikings",
        "author": "Ragnar",
        "publicationYear": 1950,
        "isbn": "12345785"
```

2)Get Book by ID

Retrieves details of a book by its ID
Method: GET
URL: http://localhost:8080/api/books/{id}

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com



GET http://localhost:8080/api/books/5

Type: Basic Auth

The authorization header will be automatically generated when you send the request. Learn

Username: admin

Password: secret

Body Cookies (1) Headers (14) Test Results

Status: 200 OK Time: 226 ms Size: 516 B

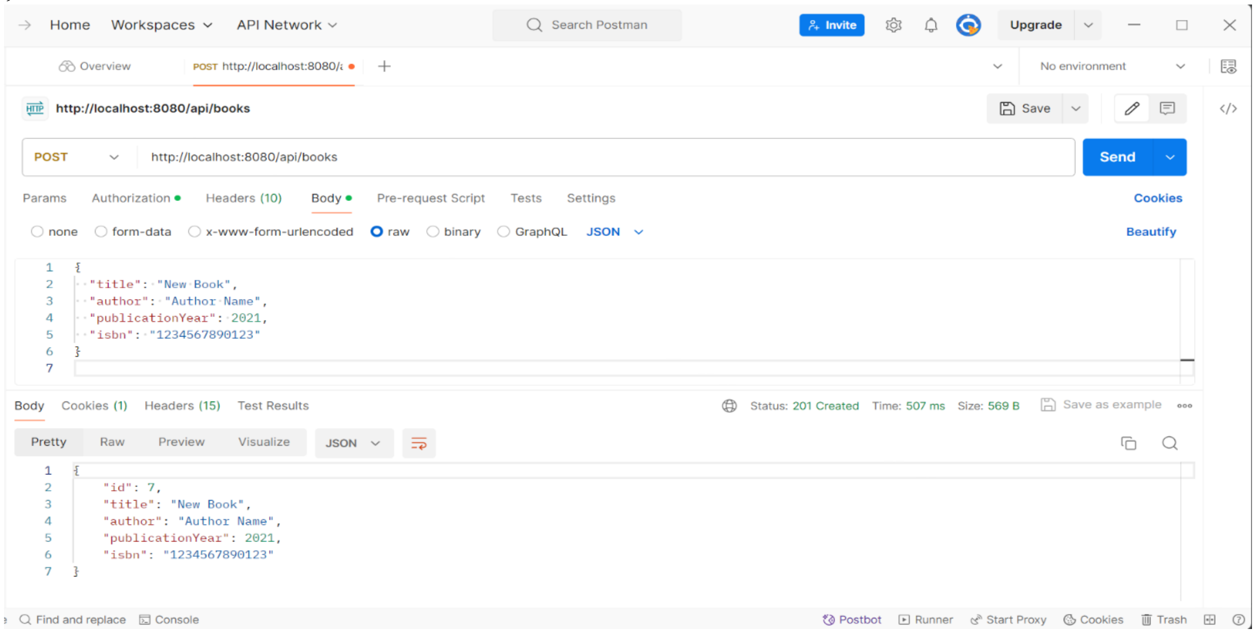
JSON

```
{
    "id": 5,
    "title": "Game of Thrones",
    "author": "Starks",
    "publicationYear": 1970,
    "isbn": "12387907"
}
```

3) Create Book

Creates an Object when invoked.
Method: POST
URL: http://localhost:8080/api/books
Body: (JSON) send the object request in JSON format as shown below

```
  {
 "title": "New Book Title",
 "author": "Author Name",
 "publicationYear": 2021,
 "isbn": "ISBN Number"
}
```



POST http://localhost:8080/api/books

none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
  "title": "New Book",
  "author": "Author Name",
  "publicationYear": 2021,
  "isbn": "1234567890123"
}
```

Body Cookies (1) Headers (15) Test Results

Status: 201 Created Time: 507 ms Size: 569 B

JSON

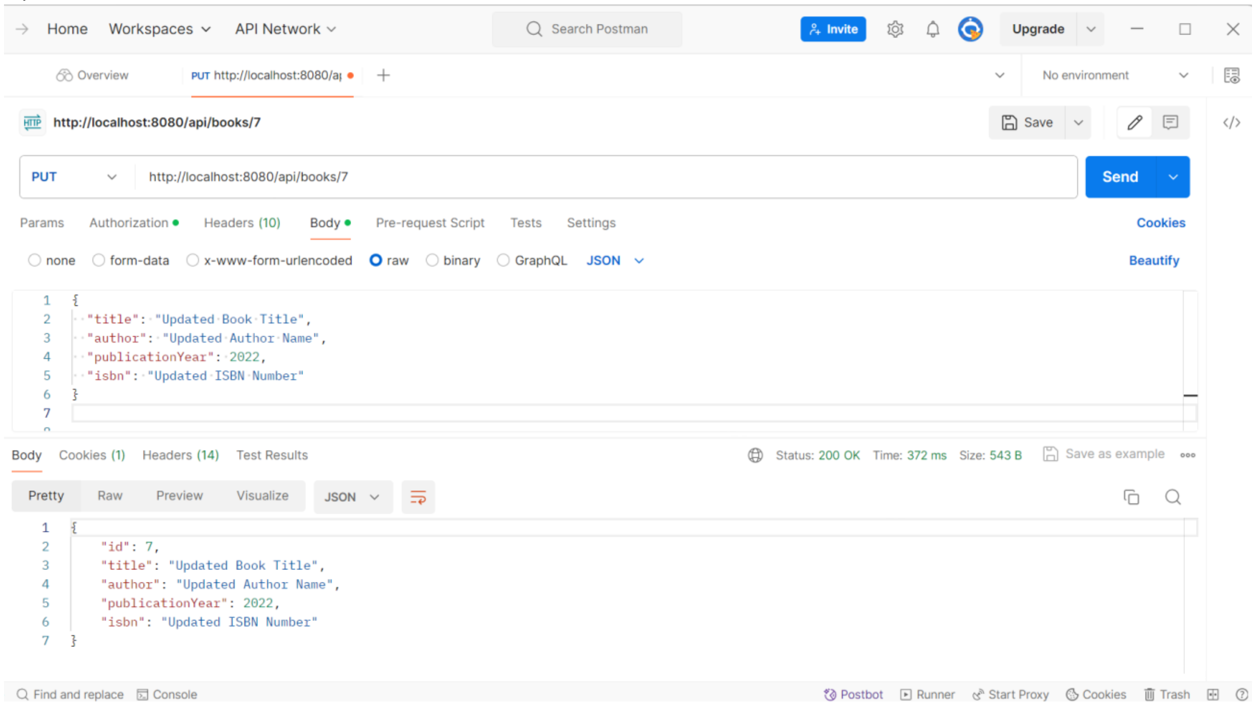
```
{
    "id": 7,
    "title": "New Book",
    "author": "Author Name",
    "publicationYear": 2021,
    "isbn": "1234567890123"
}
```

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

4) Update Book

Updates the details of any book retrieved by ID.
Method: PUT
URL: http://localhost:8080/api/books/{id}
Body: (JSON)

```
  {
 "title": "Updated Book Title",
 "author": "Updated Author Name",
 "publicationYear": 2022,
 "isbn": "Updated ISBN Number"
 }
```



http://localhost:8080/api/books/7

PUT http://localhost:8080/api/books/7

```
{
··"title":·"Updated·Book·Title",
··"author":·"Updated·Author·Name",
··"publicationYear":·2022,
··"isbn":·"Updated·ISBN·Number"
}
```

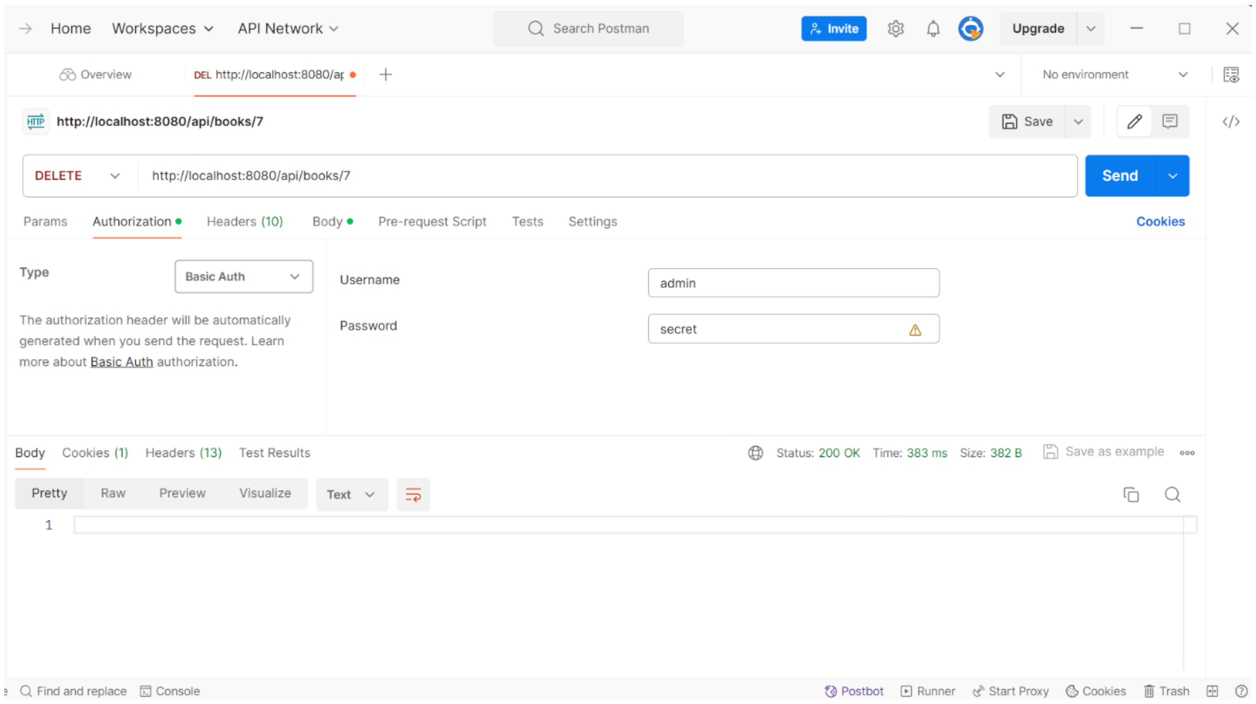
Status: 200 OK Time: 372 ms Size: 543 B

```
{
    "id": 7,
    "title": "Updated Book Title",
    "author": "Updated Author Name",
    "publicationYear": 2022,
    "isbn": "Updated ISBN Number"
}
```

5) Delete Book

Removes a book from the library.
Method: DELETE
URL: http://localhost:8080/api/books/{id}

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com



DEL http://localhost:8080/api/books/7

DELETE http://localhost:8080/api/books/7

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

Type: Basic Auth

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

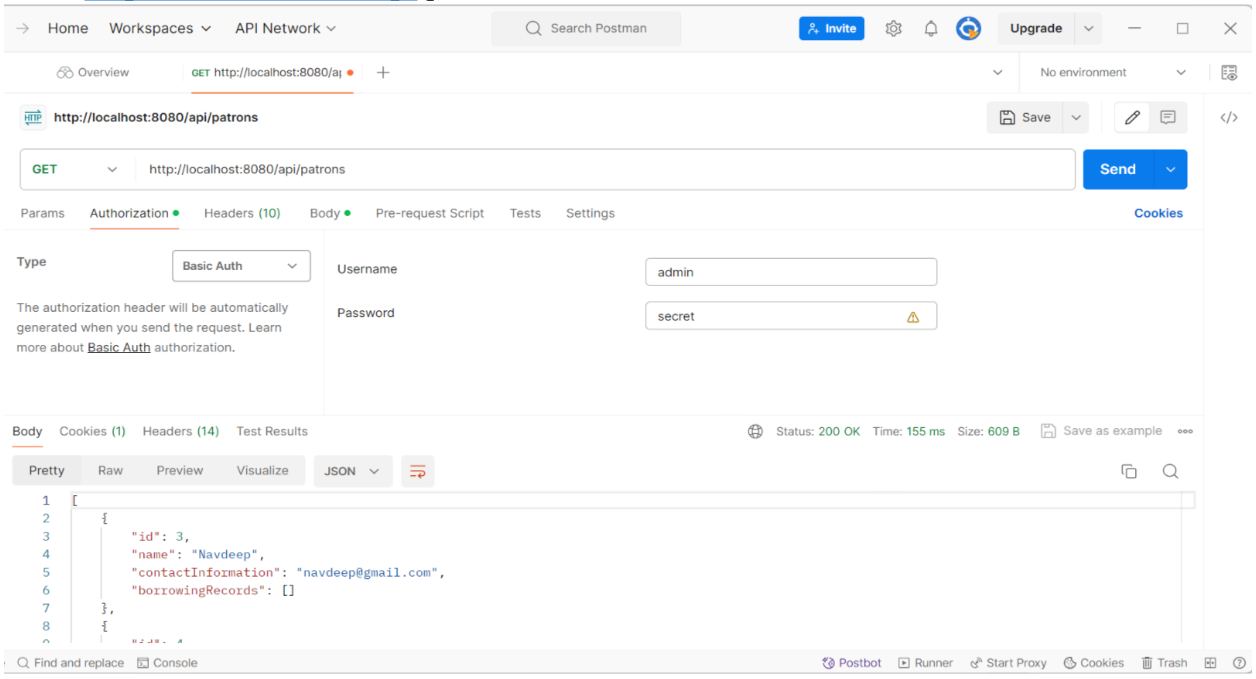
Username: admin
Password: secret

Body | Cookies (1) | Headers (13) | Test Results
Status: 200 OK Time: 383 ms Size: 382 B

Public Endpoints for Patron Entity

1) 1. Get All Patrons

To retrieve a list of all patrons.
Method: GET
URL: http://localhost:8080/api/patrons



GET http://localhost:8080/api/patrons

GET http://localhost:8080/api/patrons

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

Type: Basic Auth

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

Username: admin
Password: secret

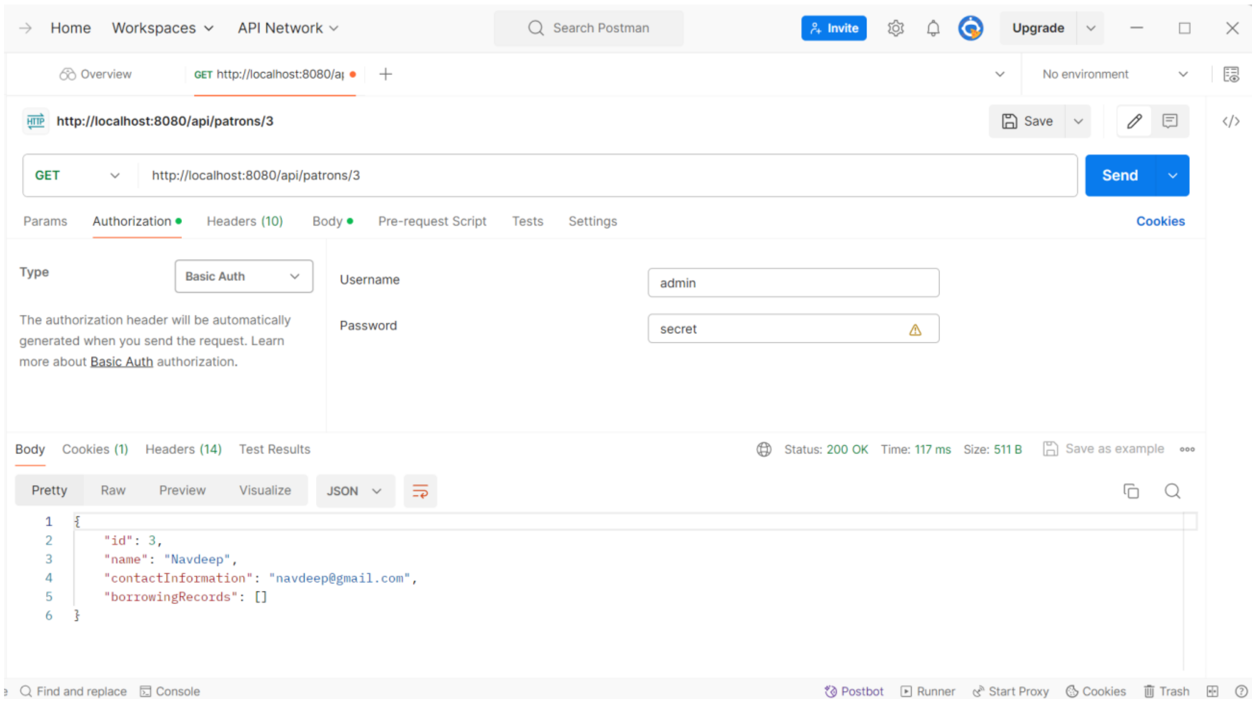
Body | Cookies (1) | Headers (14) | Test Results
Status: 200 OK Time: 155 ms Size: 609 B

```
[
    {
        "id": 3,
        "name": "Navdeep",
        "contactInformation": "navdeep@gmail.com",
        "borrowingRecords": []
    },
    {
```

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

2) Get Patron by ID

To fetch a specific patron by their unique ID.
Method: GET
URL: http://localhost:8080/api/patrons/{id}
Replace {id} with the actual ID of the patron.



GET http://localhost:8080/api/patrons/3

Authorization — Type: Basic Auth

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

Username: admin
Password: secret

Status: 200 OK Time: 117 ms Size: 511 B

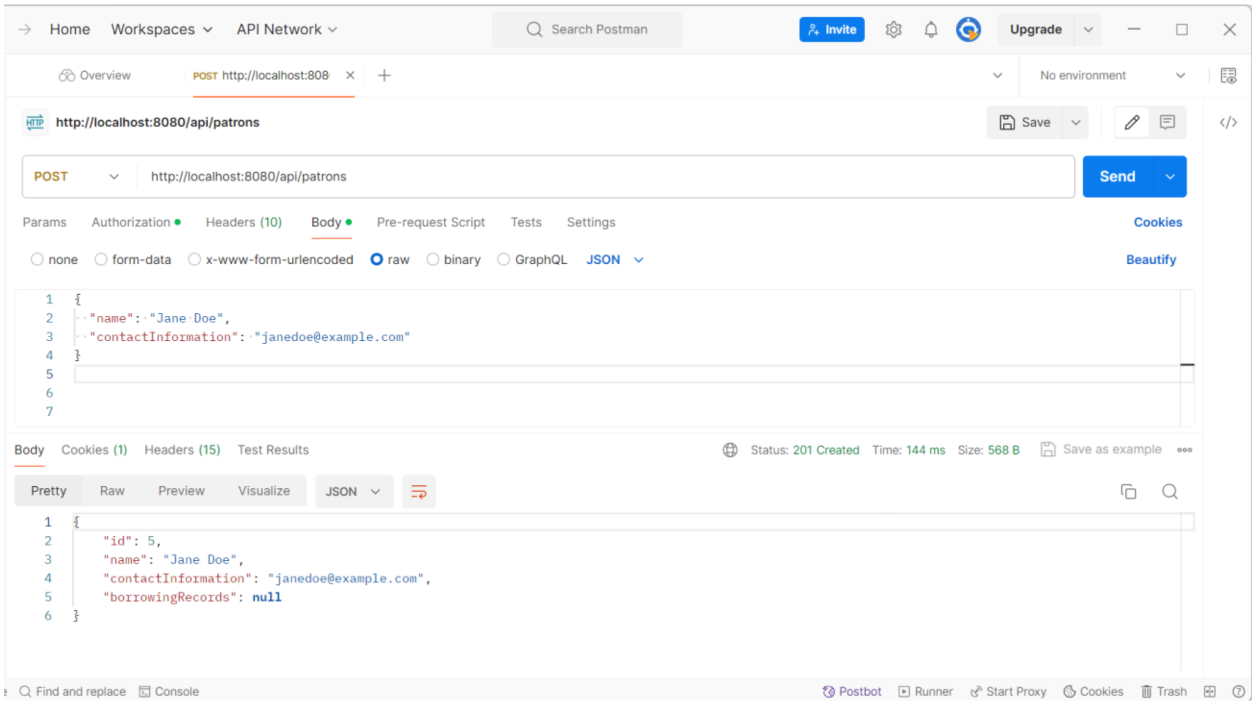
```
{
    "id": 3,
    "name": "Navdeep",
    "contactInformation": "navdeep@gmail.com",
    "borrowingRecords": []
}
```

3) Create a New Patron

To add a new patron to the library system.
Method: POST
URL: http://localhost:8080/api/patrons
Body (application/json):

```
 {
"name": "Jane Doe",
"contactInformation": "janedoe@example.com"
}
```

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com



POST http://localhost:8080/api/patrons

Body: raw JSON

```
{
  "name": "Jane Doe",
  "contactInformation": "janedoe@example.com"
}
```

Body Cookies (1) Headers (15) Test Results

Status: 201 Created Time: 144 ms Size: 568 B

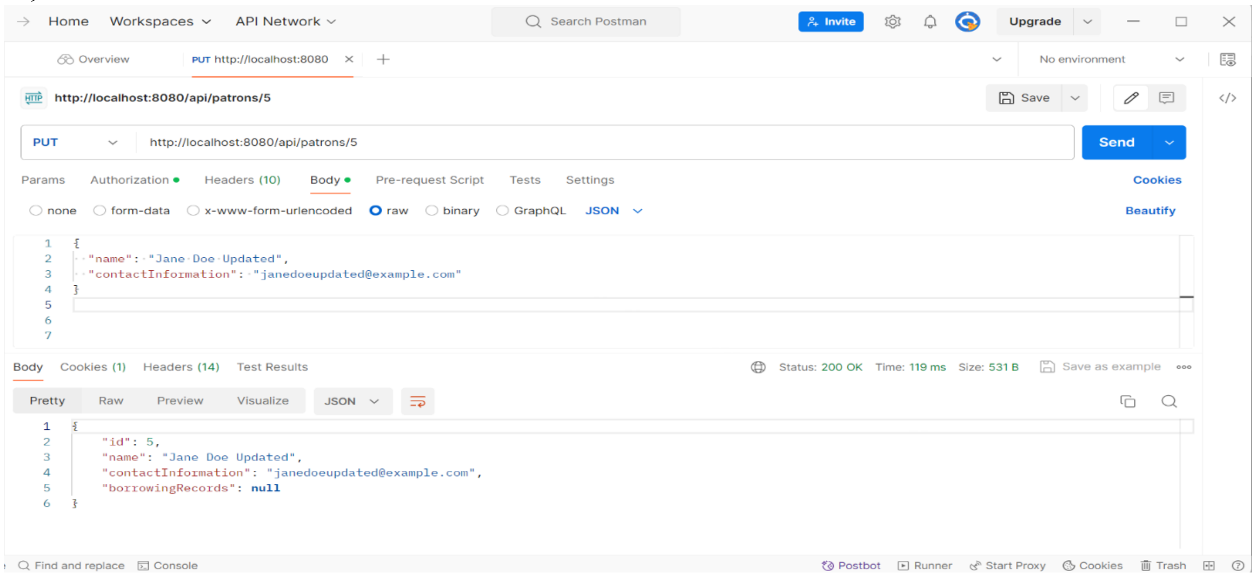
```
{
    "id": 5,
    "name": "Jane Doe",
    "contactInformation": "janedoe@example.com",
    "borrowingRecords": null
}
```

4) Update an Existing Patron

To modify the details of an existing patron.

Method: PUT
URL: http://localhost:8080/api/patrons/{id}
Replace {id} with the patron ID you wish to update.

```
 {
"name": "Jane Doe Updated",
"contactInformation": "janedoeupdated@example.com"
 }
```



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

Body: raw JSON

```
{
  "name": "Jane Doe Updated",
  "contactInformation": "janedoeupdated@example.com"
}
```

Body Cookies (1) Headers (14) Test Results

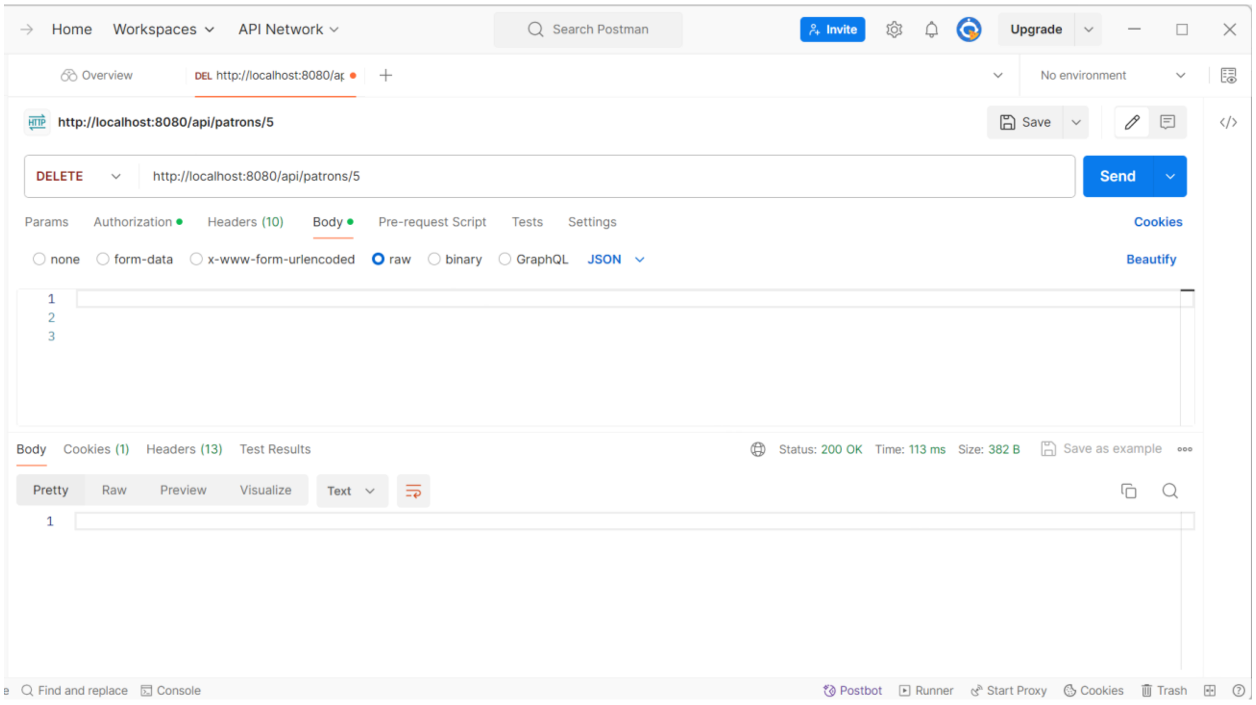
Status: 200 OK Time: 119 ms Size: 531 B

```
{
    "id": 5,
    "name": "Jane Doe Updated",
    "contactInformation": "janedoeupdated@example.com",
    "borrowingRecords": null
}
```

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

5) Delete a Patron
To remove a patron from the system.

Method: DELETE
URL: http://localhost:8080/api/api/patrons/{id}
Replace {id} with the patron ID you wish to delete.



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

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

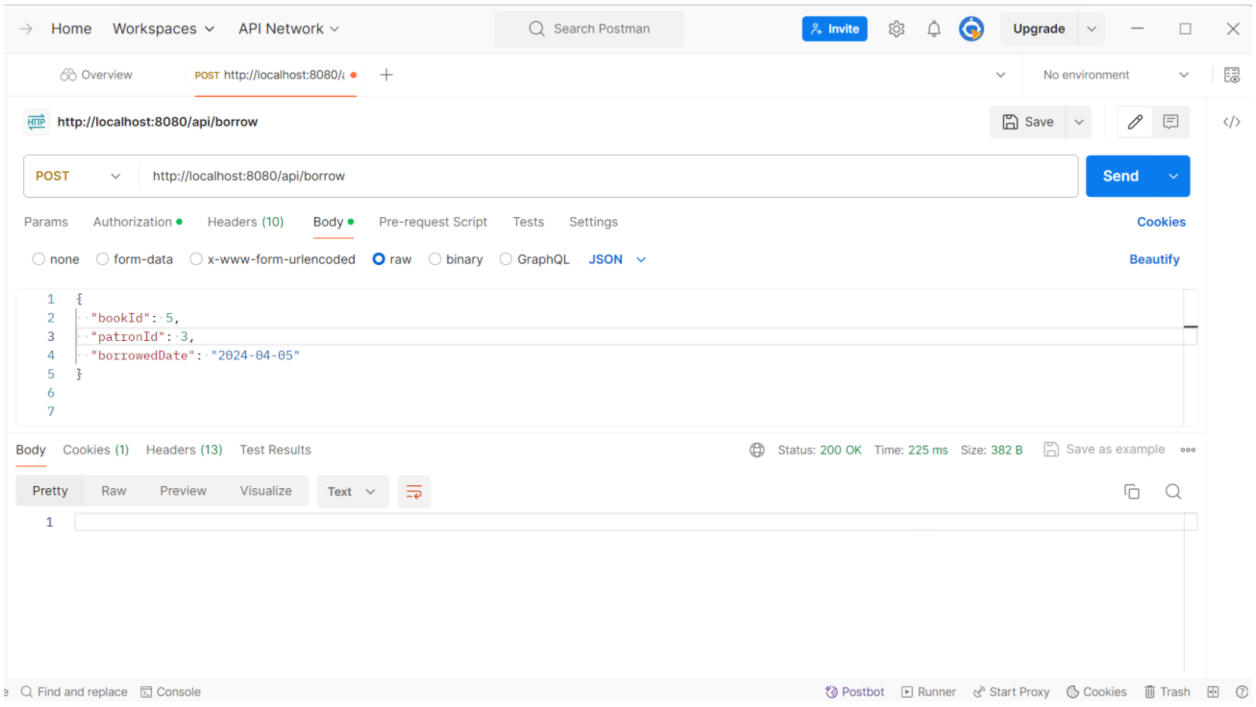
Status: 200 OK Time: 113 ms Size: 382 B

<u>Public Endpoints for Borrowing Record Entity</u>

1) **Borrow a Book**
   Creates a record in Borrowing record table with patron ID , book ID and Borrowed date.
   Method: POST
   URL: http://localhost:8080/api/borrow
   Body (example JSON):

```
   {
 "bookId": 5,
 "patronId": 3,
 "borrowedDate": "2024-04-05"
   }
```

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com



POST http://localhost:8080/api/borrow

Body: raw JSON

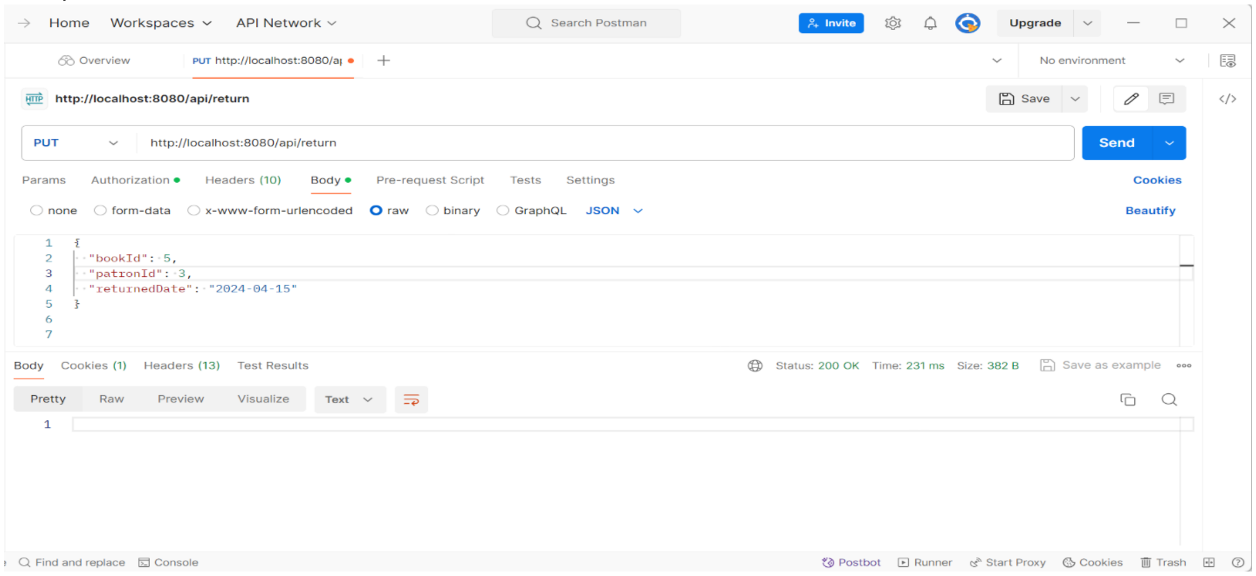
```
{
  "bookId": 5,
  "patronId": 3,
  "borrowedDate": "2024-04-05"
}
```

Status: 200 OK Time: 225 ms Size: 382 B

2) Return a Book

Updates the return date field in the borrowing record by matching patron id and book id.
Method: PUT
URL: http://localhost:8080/api/return
Body (example JSON):

```
    {
 "bookId": 5,
 "patronId": 3,
 "returnedDate": "2024-04-15"
    }
```



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

Body: raw JSON

```
{
  "bookId": 5,
  "patronId": 3,
  "returnedDate": "2024-04-15"
}
```

Status: 200 OK Time: 231 ms Size: 382 B

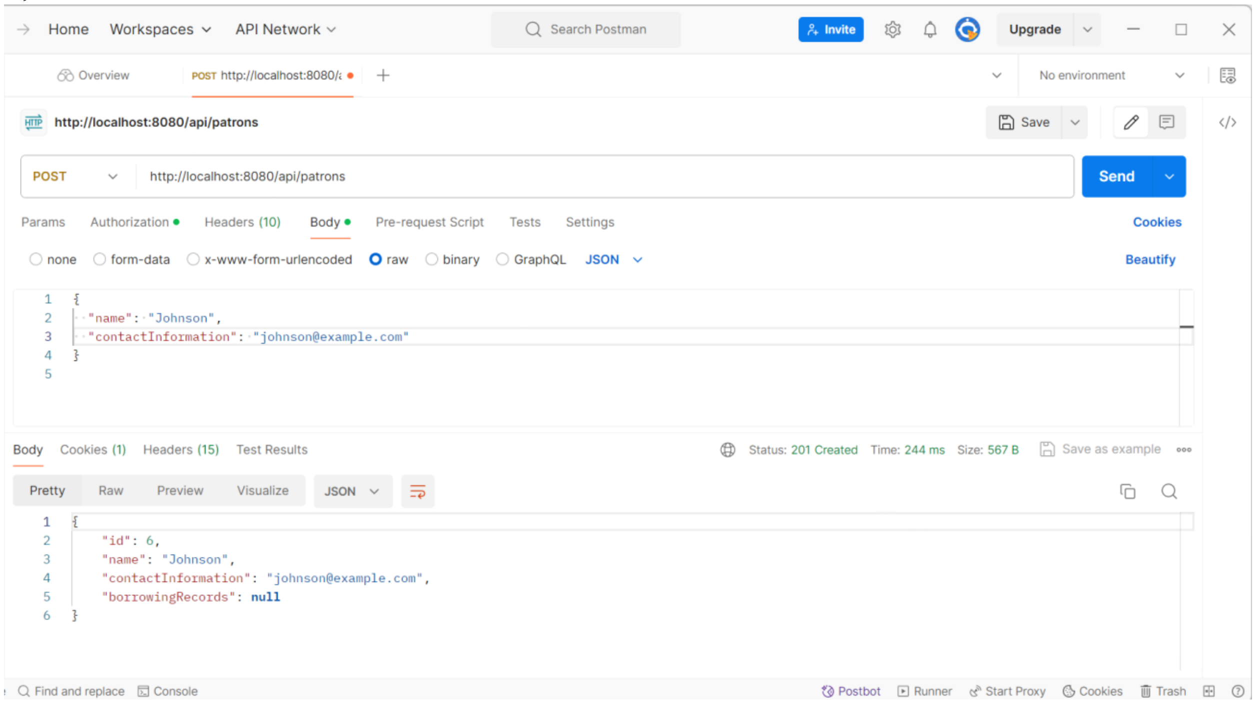
Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

WorkFlow:

To demonstrate the workflow of the Library Management application using Postman, let's follow a sequence of operations that showcase how to interact with the system. This workflow will include creating a patron, adding a book, borrowing it, and returning it. Each step will be a separate request in Postman and we will see the results in the MYSQL database.

Step 1: Firstly, We will create a patron using the post method mentioned above in the document for accessing endpoints for the patron entity. We will use details as follows

```
{
"name": "Johnson",
"contactInformation": "johnson@example.com"
}
```



http://localhost:8080/api/patrons

POST http://localhost:8080/api/patrons

Body: none form-data x-www-form-urlencoded [✓ raw] binary GraphQL JSON

```
{
  "name": "Johnson",
  "contactInformation": "johnson@example.com"
}
```

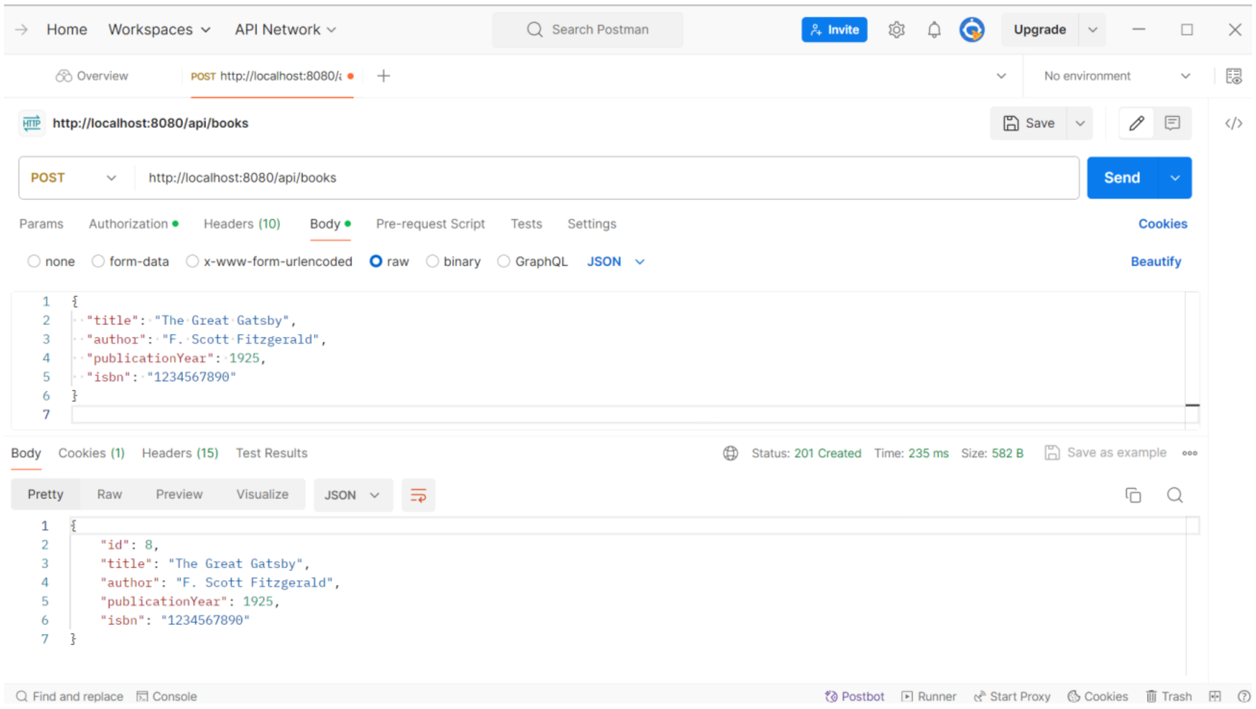
Status: 201 Created Time: 244 ms Size: 567 B

```
{
    "id": 6,
    "name": "Johnson",
    "contactInformation": "johnson@example.com",
    "borrowingRecords": null
}
```

Step 2: Now we will create a book entity using post method mentioned above in document so that patron(johnson) can borrow the book. Book details are as follows:

```
{
"title": "The Great Gatsby",
"author": "F. Scott Fitzgerald",
"publicationYear": 1925,
"isbn": "1234567890"
}
```

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com


POST http://localhost:8080/api/books

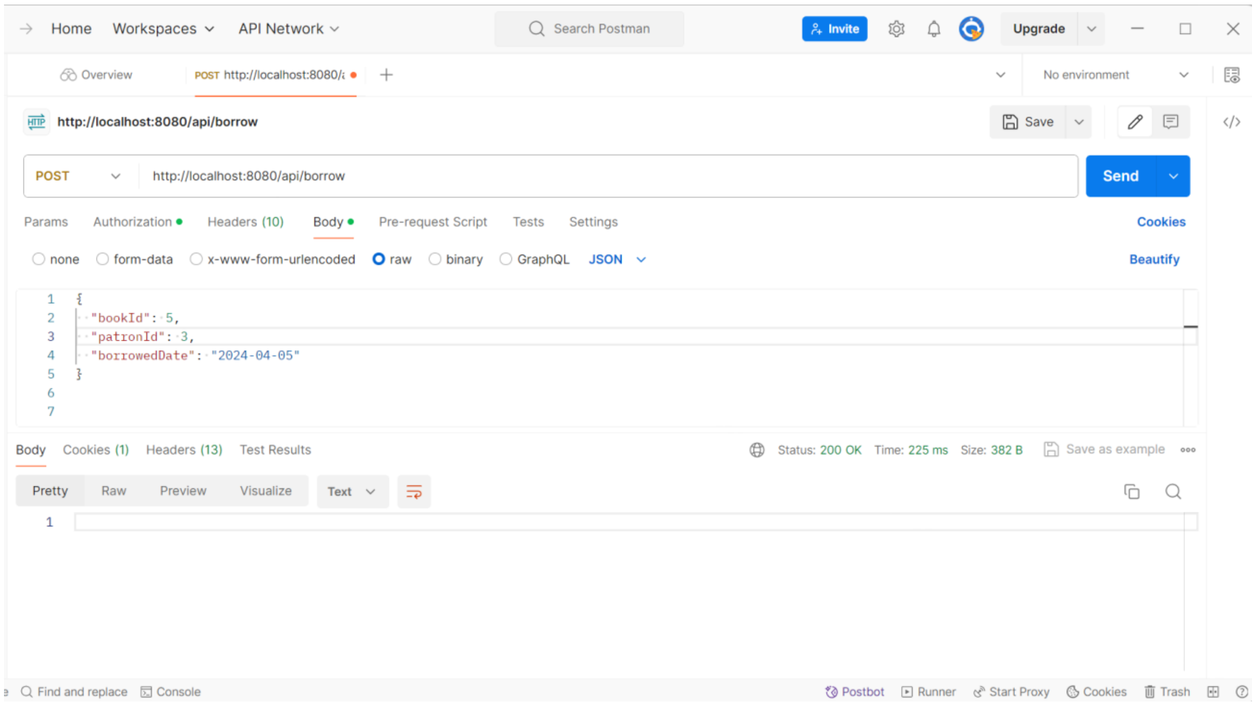
Body (raw, JSON):

```
{
  "title": "The Great Gatsby",
  "author": "F. Scott Fitzgerald",
  "publicationYear": 1925,
  "isbn": "1234567890"
}
```

Status: 201 Created Time: 235 ms Size: 582 B

```
{
    "id": 8,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "publicationYear": 1925,
    "isbn": "1234567890"
}
```

Step 3: Now let us assume patron Johnson is trying to borrow a book named The Great Gatsby. Now we will use post method in borrowing record with details as follows where I specify bookID, patronID, and borrowed date.


POST http://localhost:8080/api/borrow

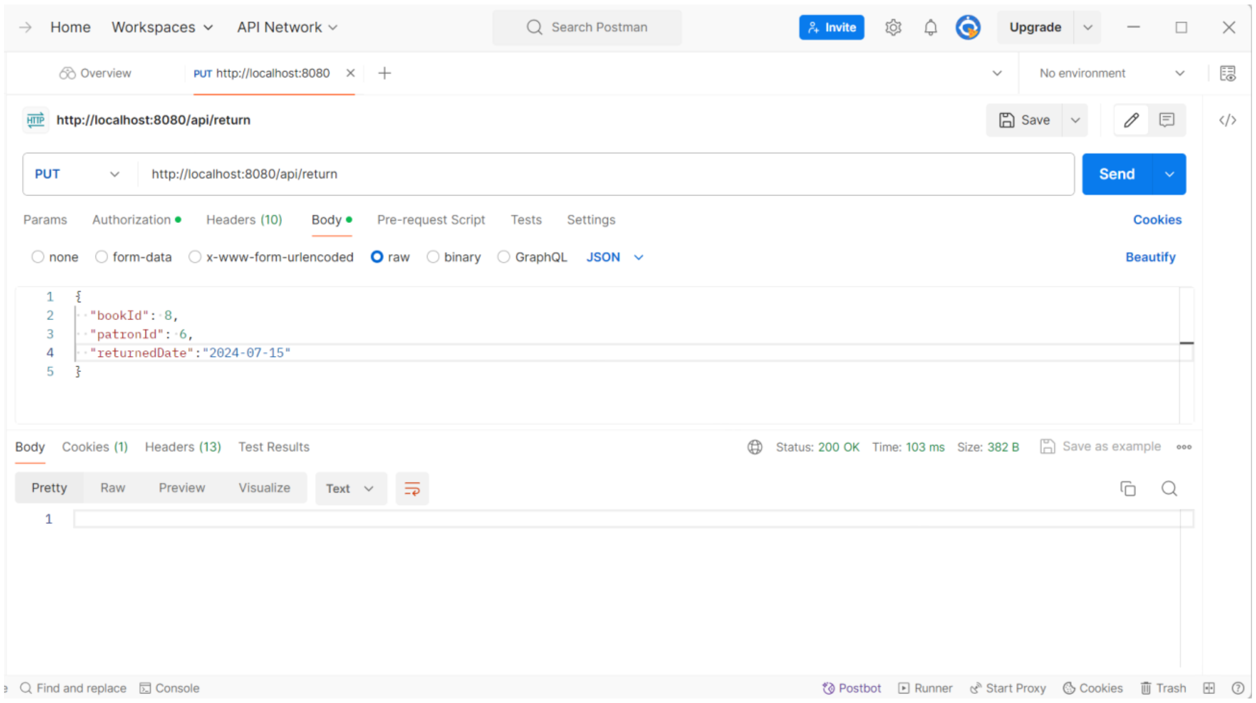
Body (raw, JSON):

```
{
  "bookId": 5,
  "patronId": 3,
  "borrowedDate": "2024-04-05"
}
```

Status: 200 OK Time: 225 ms Size: 382 B

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

Step 4: Now Johnson wants to return the book and we need to use the put method in borrow record as mentioned above as we just want to update the return date field and don't want to create a new field for this.



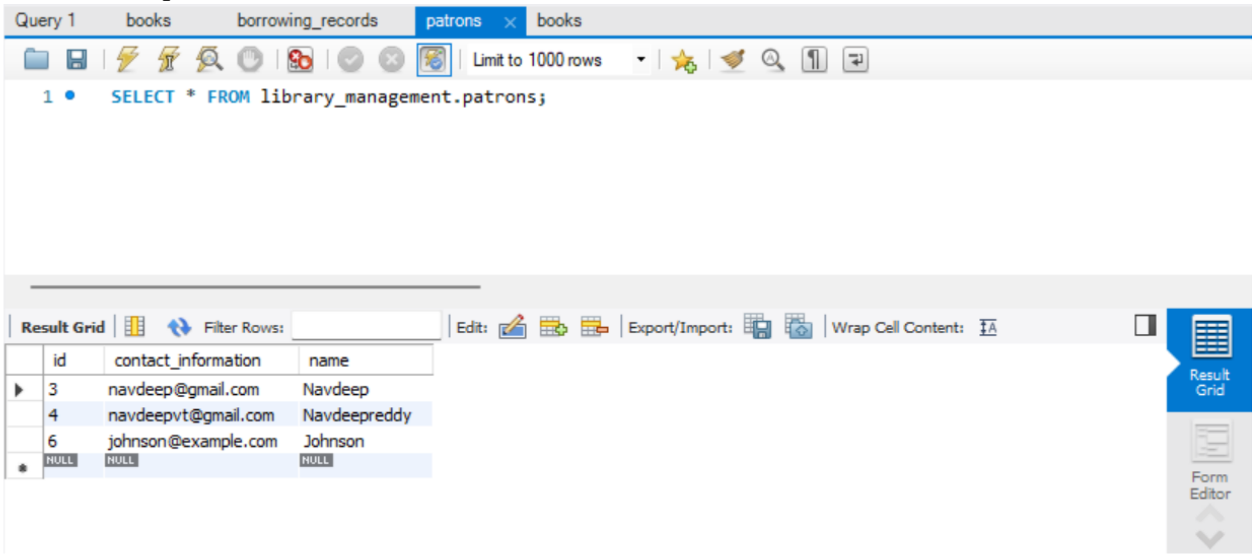
PUT http://localhost:8080/api/return

```
{
  "bookId": 8,
  "patronId": 6,
  "returnedDate":"2024-07-15"
}
```

Status: 200 OK Time: 103 ms Size: 382 B

Results:

Now lets see results in the MYSQL database.
Lets see the patron table in database to check if it is inserted. Johnson is inserted with id 6

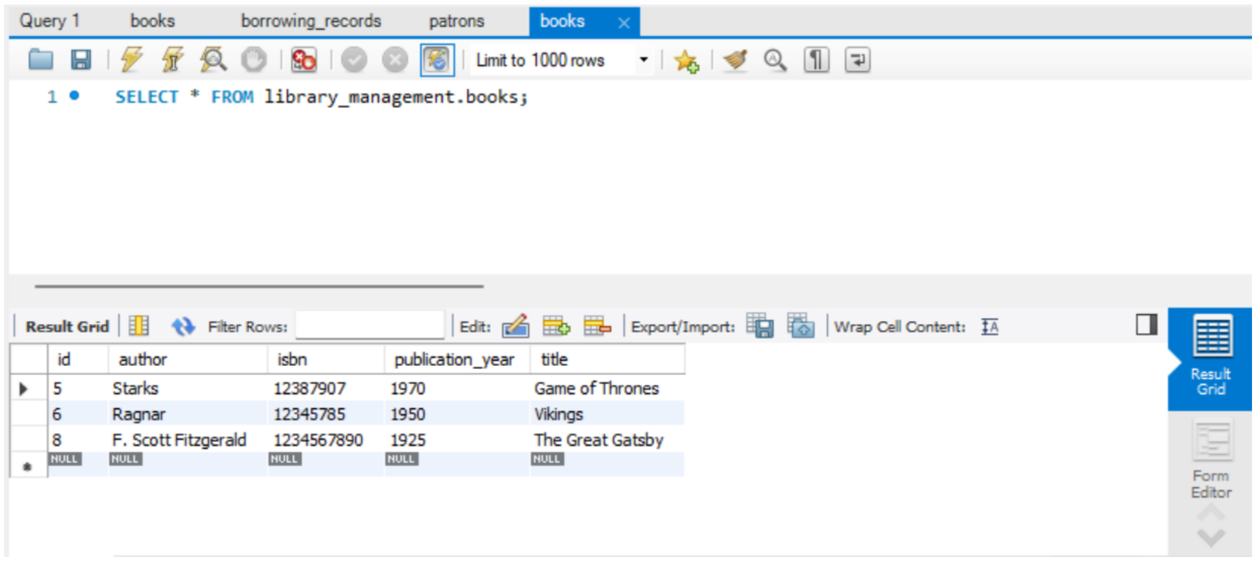


```
SELECT * FROM library_management.patrons;
```

id	contact_information	name
3	navdeep@gmail.com	Navdeep
4	navdeepvt@gmail.com	Navdeepreddy
6	johnson@example.com	Johnson
NULL	NULL	NULL

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

Now, Lets see Book Table in Database if it is inserted. Book is inserted with Id 8.

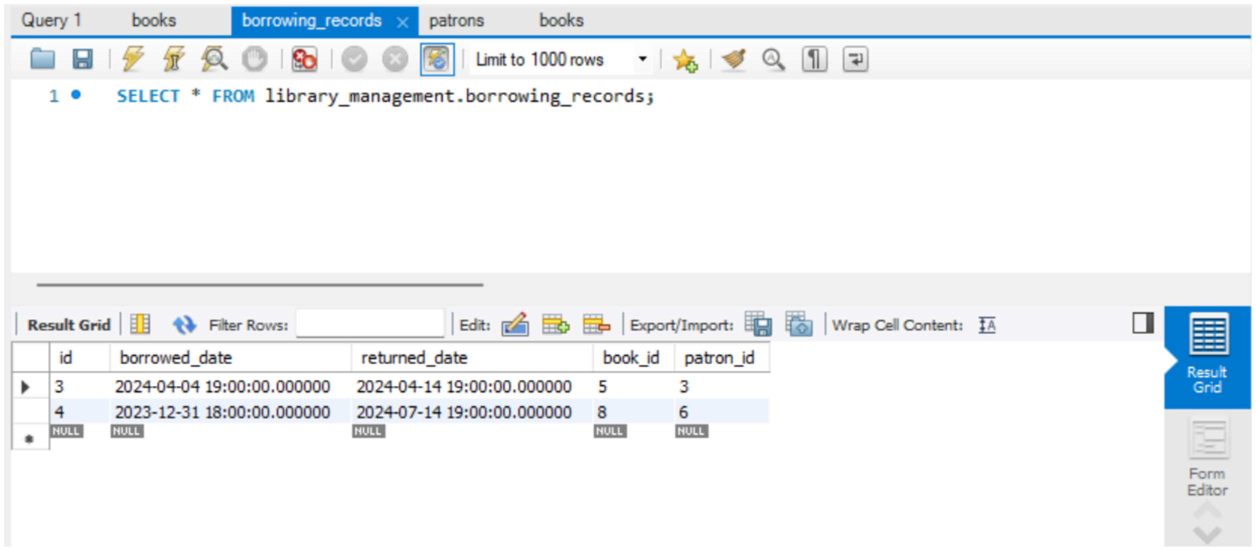


Query 1 | books | borrowing_records | patrons | books

```
SELECT * FROM library_management.books;
```

id	author	isbn	publication_year	title
5	Starks	12387907	1970	Game of Thrones
6	Ragnar	12345785	1950	Vikings
8	F. Scott Fitzgerald	1234567890	1925	The Great Gatsby
NULL	NULL	NULL	NULL	NULL

We can see that in the Borrowing Record Johnson with ID 6 is matched to the book 8 with borrowed date and returned date we mentioned in the endpoints in workflow.



Query 1 | books | borrowing_records | patrons | books

```
SELECT * FROM library_management.borrowing_records;
```

id	borrowed_date	returned_date	book_id	patron_id
3	2024-04-04 19:00:00.000000	2024-04-14 19:00:00.000000	5	3
4	2023-12-31 18:00:00.000000	2024-07-14 19:00:00.000000	8	6
NULL	NULL	NULL	NULL	NULL

Author: Navdeep Reddy Vitta
Mail: vittanavdeep123@gmail.com

<u>Application Features:</u>

Validation and Error Handling:
Implemented custom exception handling models such as PatronNotFound exception, BookNotFound exception and more which are handled by GlobalExceptionHandler (@ControllerAdvice). Validated input and data fields using @NotNull, @Validate, etc.

Security:
Implemented basic authentication using spring security feature for authentication of endpoints. Configured Web security module to disable CRSF token so that we can access post methods.

Aspects:
Desgined LoggingAspect using Aspect-Oriented Programming (AOP) to enhance the Library Management application by providing detailed logging capabilities, particularly focusing on performance metrics. It intercepts all method calls within the BookController class, measuring and logging the execution time of these methods. This aspect uses @Around advice to wrap around the method execution, allowing it to log the method name and its execution time to the **application.log** file, aiding in performance monitoring and troubleshooting without intruding into the business logic.

Caching:
Implemented caching mechanism in book service and patron service layers using @cacheble, @cachePut, and @cacheEvict for respective operations to enhance the system performance.

Transaction Management:
Implemented declarative transaction management using Spring's @Transactional annotation in service layers to ensure data integrity during critical operations.

Testing:
Designed 29 unit test cases using JUnit and Mockito to test various functionalities and behaviors of the rest endpoints in the application. Written Test Modules for each service and Controller layer of book, patron, and borrowing record.