

# Library Management System API with Spring Boot

This document outlines the design and development of a Library Management System API using Spring Boot.

## Project Goals

- Build a RESTful API for managing books, patrons, and borrowing records in a library.
- Provide functionalities for librarians to interact with the library data.

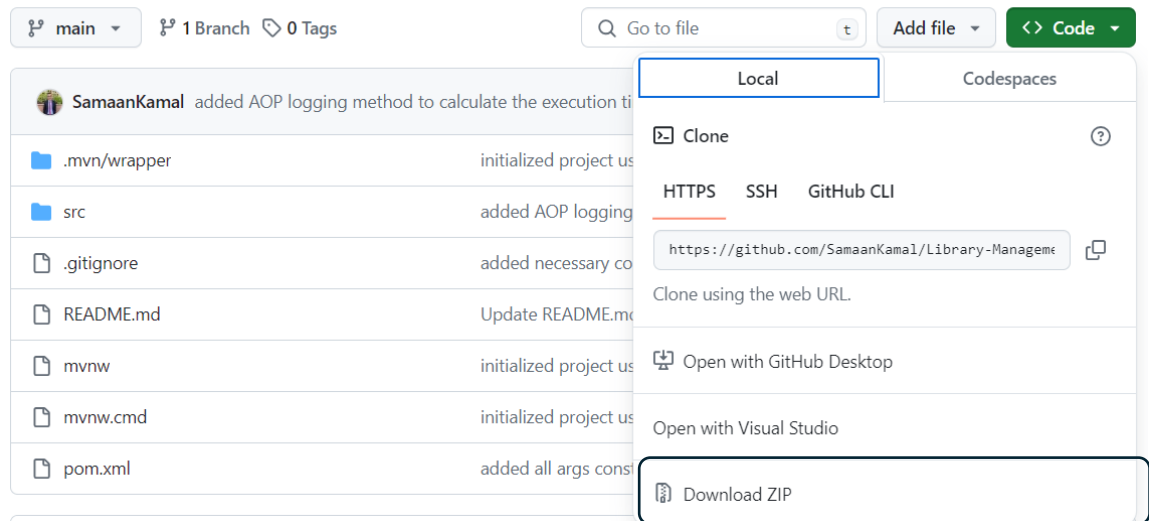
## How To Run the Project

### 1. Clone the repository or Download ZIP File:

<https://github.com/SamaanKamal/Library-Management-System-Backend-Spring-Boot>

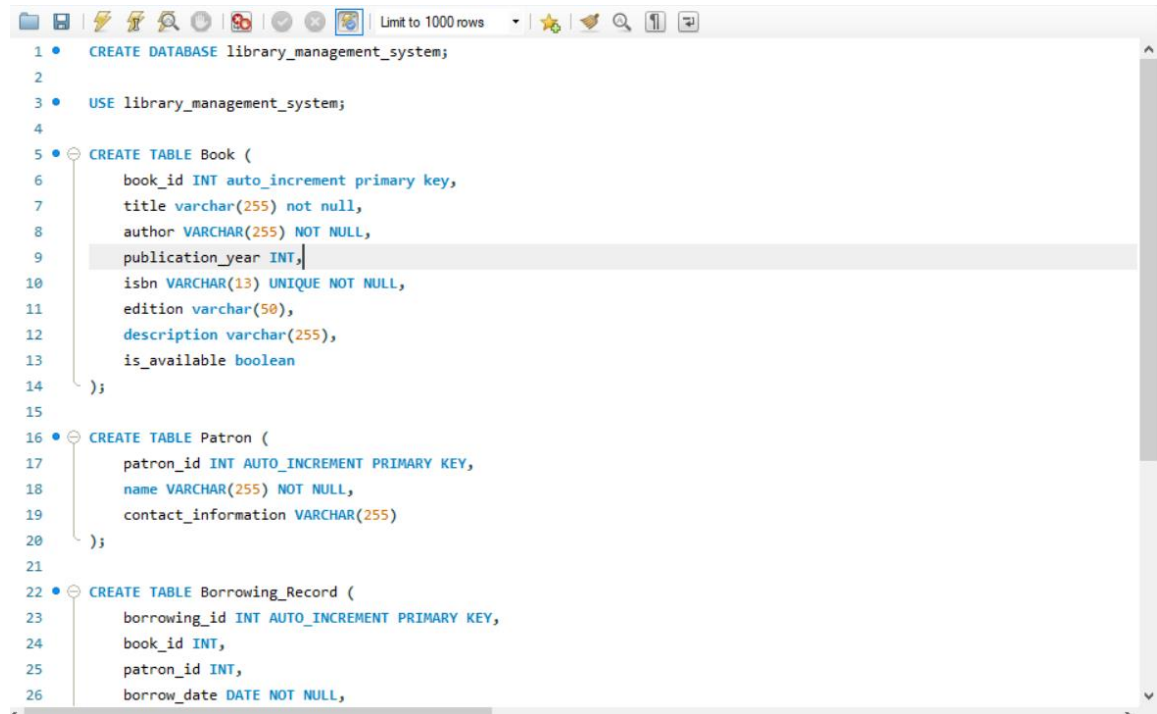
```
git clone https://github.com/SamaanKamal/Library-Management-System-Backend-Spring-Boot.git
```

or



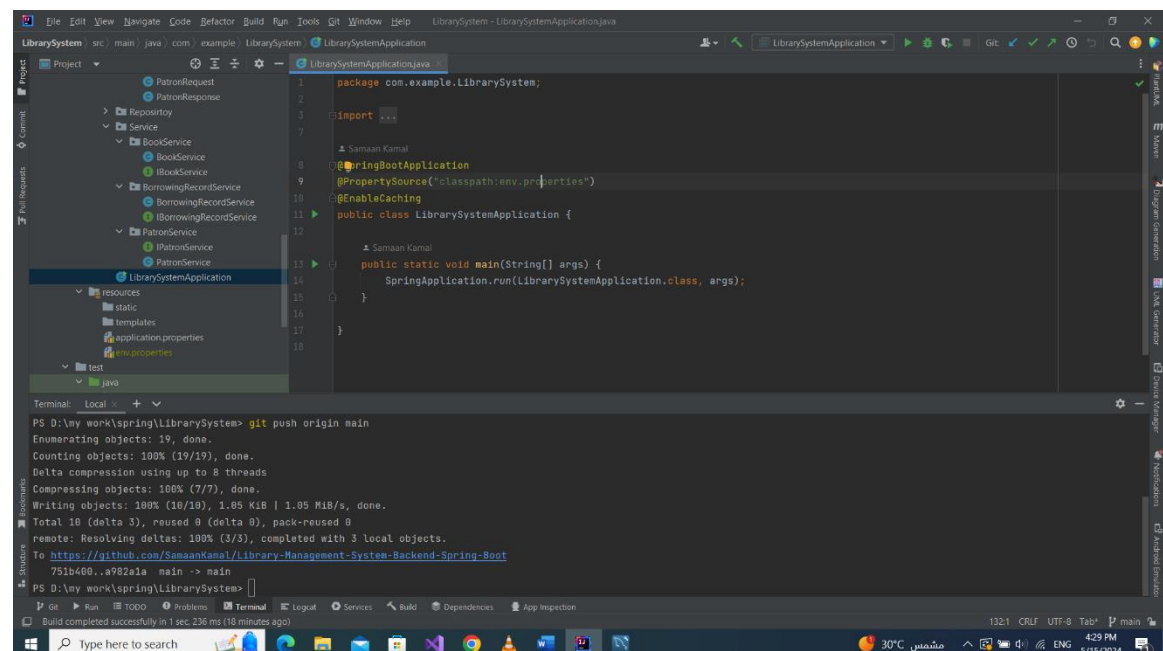
### 2. cd SimpleTaskManagement or using the folder itself.

### 3. Open the Database Folder and Run the SQL script in it using MySQL workbench.



```
1 • CREATE DATABASE library_management_system;
2
3 • USE library_management_system;
4
5 • CREATE TABLE Book (
6     book_id INT auto_increment primary key,
7     title varchar(255) not null,
8     author VARCHAR(255) NOT NULL,
9     publication_year INT,
10    isbn VARCHAR(13) UNIQUE NOT NULL,
11    edition varchar(50),
12    description varchar(255),
13    is_available boolean
14 );
15
16 • CREATE TABLE Patron (
17     patron_id INT AUTO_INCREMENT PRIMARY KEY,
18     name VARCHAR(255) NOT NULL,
19     contact_information VARCHAR(255)
20 );
21
22 • CREATE TABLE Borrowing_Record (
23     borrowing_id INT AUTO_INCREMENT PRIMARY KEY,
24     book_id INT,
25     patron_id INT,
26     borrow_date DATE NOT NULL,
```

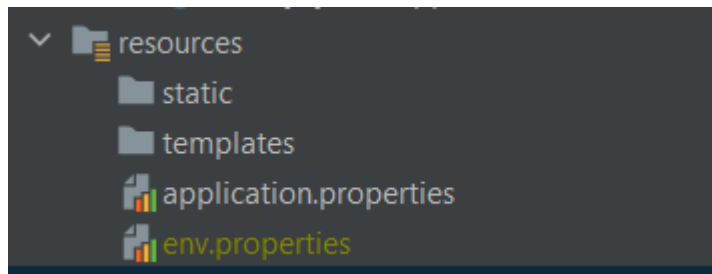
### 4. Open the Project using IDE supports maven and java.



```
1 package com.example.LibrarySystem;
2
3 import ...
4
5 @SpringBootApplication
6 @PropertySource("classpath:env.properties")
7 @EnableCaching
8 public class LibrarySystemApplication {
9
10     @Samaan Kamal
11     public static void main(String[] args) {
12         SpringApplication.run(LibrarySystemApplication.class, args);
13     }
14 }
15
16
17
18
```

```
Terminal: Local x +
PS D:\my work\spring\LibrarySystem> git push origin main
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 1.05 KiB | 1.05 MiB/s, done.
Total 10 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/SamaanKamal/Library-Management-System-Backend-Spring-Boot
751b460..a982a1a main -> main
PS D:\my work\spring\LibrarySystem>
```

**5. create env.properties file next to application.properties**



**6. open env.properties file and the following pattern:**

```
DB_DATABASE=jdbc:mysql://localhost:3306/[your schema name]
DB_USER=[your user database username]
DB_PASSWORD=[your user database password]
SECRET_KEY=[your 256 bit Secret Key]
```

**7. Remove the Brackets and what is inside it with the correct data you have 😊**

**8. That is it You can now the Project**

## Usage

```
Navigate to the project directory
Build the project
Run the application
Access the application at http://localhost:8080 or any port you configured
test the API endpoints with postman.
```

## API Endpoints

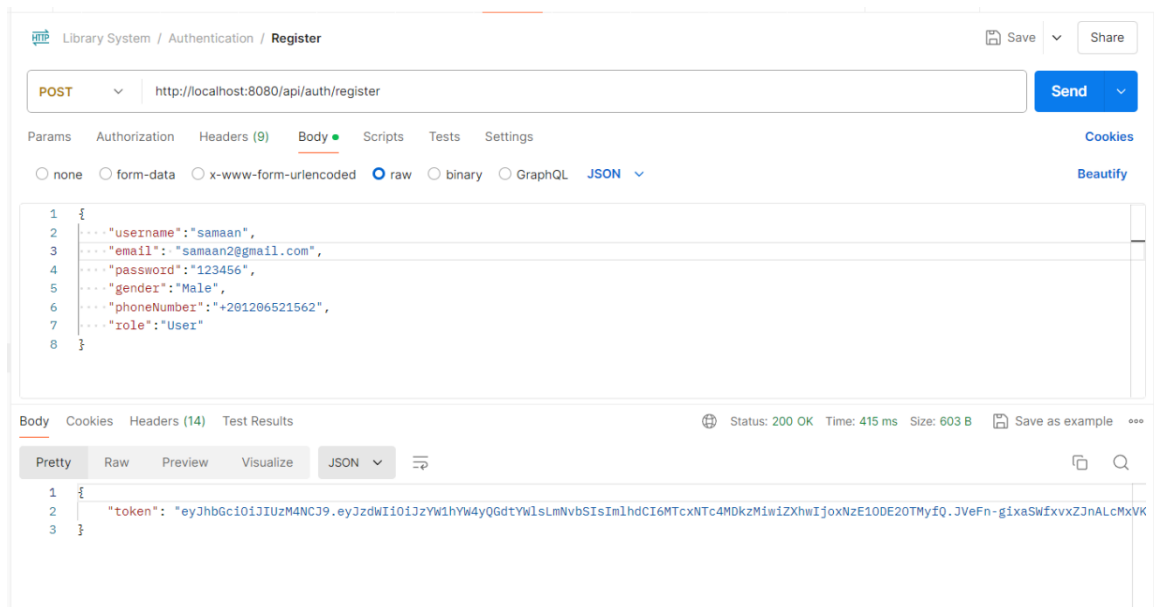
The API will expose functionalities through RESTful endpoints:

First of all, you will need to register with your information to access any endpoint

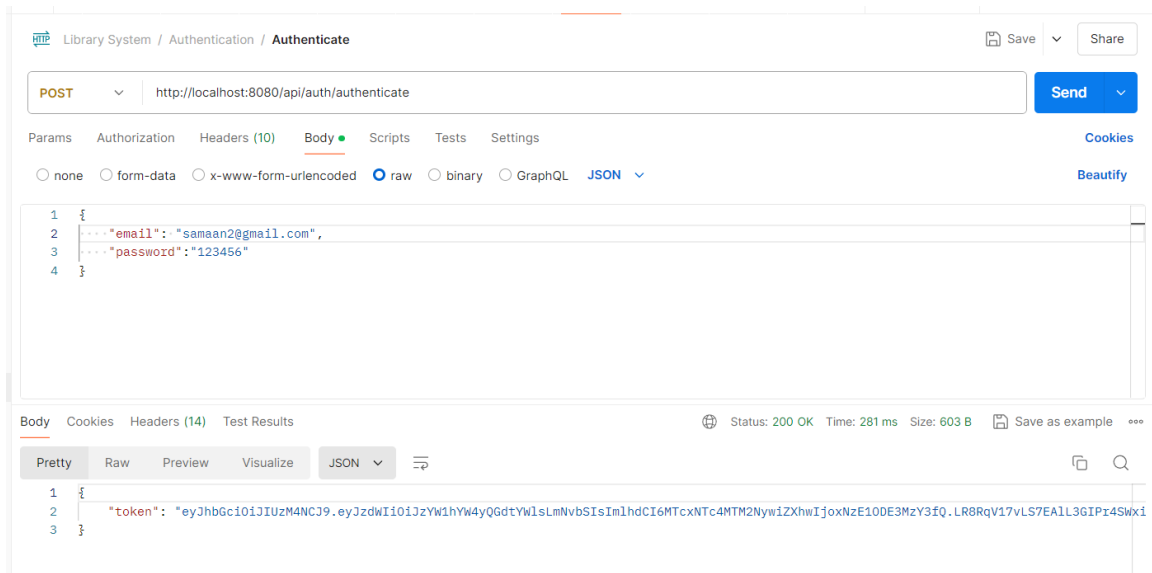
So

Authentication:

- **Post: /api/auth/register:** you will need to send your username and password and gender and phone number with “User” Role and a UNIQUE email address every time you register once you provided this information you will get JWT token for 10 hours to allow you to access any other endpoint



- **Post: /api/auth/authenticate:** you will need to send your email and password you provided before in the registration after the token expired to authenticate again and get new token to access endpoints.



## Book Management:

### Note:

- **Before you start sending any Requests to any endpoint please make sure to set the authorization header with “Bearer “+ token you have (Make sure that there is a space between the two) otherwise you will get forbidden 403**
- GET: /api/books: Retrieves a list of all books, all you need to do is provide is to send the request (no data Required).

Library System / Book / Get All the Books

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

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

Key	Value	Description
Content-Type	application/json	
Authorization	Bearer eyJhbGciOiJIUzI4NCJ9.eyJzdWIiOiJzYW1hYW4...	

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 154 ms Size: 594 B Save as example

```
1 {
2   "books": [
3     {
4       "bookId": 1,
5       "title": "test Book",
6       "author": "samaan",
7       "publicationYear": 2024,
8       "isbn": "978-90",
9       "edition": "second",
10      "description": "second Test Book",
11      "available": false
12    }
13  ]
14 }
```

- GET: /api/books/{id}: Retrieves details of a specific book by its ID, here you need to provide the id in url format

Library System / Book / Get Book

GET http://localhost:8080/api/books/1

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

Key	Value	Description
Content-Type	application/json	
Authorization	Bearer eyJhbGciOiJIUzI4NCJ9.eyJzdWIiOiJzYW1hYW4...	

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 35 ms Size: 582 B Save as example

```
1 {
2   "bookId": 1,
3   "title": "test Book",
4   "author": "samaan",
5   "publicationYear": 2024,
6   "isbn": "978-90",
7   "edition": "second",
8   "description": "second Test Book",
9   "available": false
10 }
```

- **POST: /api/books:** Adds a new book to the library, here you need to provide the book information to create new book as shown(make sure the isbn is unique)

Library System / Book / Create Book

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

Send

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

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

```
1 {
2   "title": "test Book number 5",
3   "author": "samaa",
4   "publicationYear": "2024",
5   "isbn": "978-90000-10",
6   "edition": "fifth",
7   "description": "fifth Test Book"
8 }
```

Body Cookies Headers (14) Test Results Status: 201 Created Time: 30 ms Size: 599 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "bookId": 2,
3   "title": "test Book number 5",
4   "author": "samaa",
5   "publicationYear": 2024,
6   "isbn": "978-90000-10",
7   "edition": "fifth",
8   "description": "fifth Test Book",
9   "available": true
10 }
```

- **PUT: /api/books/{id}:** Updates information of an existing book, here you will need to provide the id of the book will be updated and the information that will be updated

Library System / Book / Update Book

PUT http://localhost:8080/api/books/2

Send

Params Authorization Headers (10) Body Scripts Tests Settings Cookies Beautify

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

```
1 {
2   "title": "test Book for testers",
3   "author": "samaa kamal",
4   "publicationYear": "2024",
5   "isbn": "978-909999999",
6   "edition": "third",
7   "description": "third Test Book"
8 }
```

Body Cookies Headers (14) Test Results Status: 200 OK Time: 43 ms Size: 604 B Save as example

Pretty Raw Preview Visualize JSON

```
1 {
2   "bookId": 2,
3   "title": "test Book for testers",
4   "author": "samaa kamal",
5   "publicationYear": 2024,
6   "isbn": "978-909999999",
7   "edition": "third",
8   "description": "third Test Book",
9   "available": true
10 }
```

- **DELETE:** `/api/books/{id}`: Removes a book from the library, here you need to provide the id of the book you want to delete in the url.

Library System / Book / Delete Book

DELETE `http://localhost:8080/api/books/2` **Send**

Params Authorization Headers (8) Body Scripts Tests Settings Cookies

Headers 6 hidden

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI4NCJ9.eyJzdWwiOiJzYW1hYW4...	
Key	Value	Description

Body Cookies Headers (14) Test Results

Status: 200 OK Time: 30 ms Size: 448 B Save as example

Pretty Raw Preview Visualize Text

1 Book Deleted Successfully

## Patron Management:

### The same like book endpoints

- **GET:** `/api/patrons`: Retrieves a list of all patrons.
- **GET:** `/api/patrons/{id}`: Retrieves details of a specific patron by ID.
- **POST:** `/api/patrons`: Adds a new patron to the system.
- **PUT:** `/api/patrons/{id}`: Updates information of an existing patron.
- **DELETE:** `/api/patrons/{id}`: Removes a patron from the system.

## Borrowing Management:

- **POST:** `/api/borrow/{bookId}/patron/{patronId}`: Allows a patron to borrow a book (with available check and update borrowing record), here you need to provide the id of the book and id of the patron in url in the mentioned order.

The screenshot shows a REST client interface for a "Library System". The active tab is "Borrowing Record" with a sub-tab "Borrow book". The request method is "POST" and the URL is `http://localhost:8080/api/borrow/1/patron/1`. The "Headers" tab is selected, showing two headers: "Content-Type" with value "application/json" and "Authorization" with a Bearer token. The "Body" tab is also visible, showing a JSON payload. The status bar indicates a successful response: "Status: 201 Created", "Time: 58 ms", and "Size: 740 B".

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI4NCJ9.eyJzdWwiOiJzYW1hYWw...	

```
1 {
2   "borrowingId": 1,
3   "book": {
4     "bookId": 1,
5     "title": "test Book",
6     "author": "samaa",
7     "publicationYear": 2024,
8     "isbn": "978-90",
9     "edition": "second",
10    "description": "second Test Book",
11    "available": false
12  },
13  "patron": {
14    "patronId": 1,
15    "name": "samaa Patron",
16    "contactInformation": "+201206521562"
17  },
18  "borrowDate": "2024-05-15",
19  "returnDate": null
20 }
```

- **PUT:** `/api/return/{bookId}/patron/{patronId}`: Records the return of a borrowed book (update borrowing record with return date), here you need to provide the id of the book and id of the patron in url in the mentioned order.

The screenshot shows the same REST client interface, but now for a "Return book" operation. The request method is "PUT" and the URL is `http://localhost:8080/api/return/1/patron/1`. The headers are identical to the previous request. The JSON payload in the body is updated: the "available" status of the book is now "true" and the "returnDate" is set to "2024-05-15". The status bar shows a successful response: "Status: 200 OK", "Time: 53 ms", and "Size: 742 B".

Key	Value	Description
<input checked="" type="checkbox"/> Content-Type	application/json	
<input checked="" type="checkbox"/> Authorization	Bearer eyJhbGciOiJIUzI4NCJ9.eyJzdWwiOiJzYW1hYWw...	

```
1 {
2   "borrowingId": 1,
3   "book": {
4     "bookId": 1,
5     "title": "test Book",
6     "author": "samaa",
7     "publicationYear": 2024,
8     "isbn": "978-90",
9     "edition": "second",
10    "description": "second Test Book",
11    "available": true
12  },
13  "patron": {
14    "patronId": 1,
15    "name": "samaa Patron",
16    "contactInformation": "+201206521562"
17  },
18  "borrowDate": "2024-05-15",
19  "returnDate": "2024-05-15"
20 }
```



## Entities

The system will model three primary entities:

1. **Book:** Represents a book in the library with attributes like:
  - ID (unique identifier)
  - Title
  - Author(s)
  - Publication Year
  - ISBN (Unique Value)
  - Description
  - Is Available
2. **Patron:** Represents a library member with details like:
  - ID (unique identifier)
  - Name
  - Contact Information (Phone number)
3. **Borrowing Record:** Tracks the association between books and patrons, including:
  - ID (unique identifier)
  - Book (reference to Book entity)
  - Patron (reference to Patron entity)
  - Borrowed Date
  - Returned Date (initially null)

## Relationships:

- A Book can have many Borrowing Records (One-to-Many)
- A Patron can have many Borrowing Records (One-to-Many)

## Data Storage

- Utilizes MySQL database to persist book, patron, and borrowing record details.
- Establishes appropriate relationships between entities (e.g., one-to-many between books and borrowing records).

## Validation and Error Handling

- API requests will be validated for required fields, data formats, and business rules (e.g., checking book availability before borrowing).
- Exceptions are handled gracefully, returning appropriate HTTP status codes (e.g., 400 for Bad Request, 404 for Not Found) and informative error messages.

## Security

- Implemented JWT-based authorization to protect the API endpoints.

## **Aspects**

- Implemented Logging messages before and after every method runs for the controller, also if there is any exception found the afterThrowing method will be executed.
- Also there is a method for some methods in the service that calculates the execution time for these methods.

## **Caching**

- Used Spring Caching with enabling it at first, used annotations like Cacheable for Caching the fetched data from database, CachePut for editing and updating and CacheEvict for deleting using id in the Cache.

## **Testing**

- Wrote Unit tests to validate the functionality of individual API endpoints using frameworks like JUnit, Mockito, or SpringBootTest.
- Every method is annotated with Test Annotation and provide mocking objects to test the endpoints in some cases not like bad request and successful and internal server error on the controller level.