Book Library API

Introduction

The Book Library API is a microservices-based application designed to streamline the management of a library's book inventory and subscriber records. This API provides a set of RESTful endpoints that allow librarians to efficiently handle book availability, manage subscriptions, and ensure smooth operations within the library.

The system is built using Spring Boot and follows a microservices architecture, which enhances scalability, maintainability, and flexibility. Each microservice is responsible for a specific domain within the library system, ensuring a clear separation of concerns and easier management.

Key components of the Book Library API include:

- **Service Discovery and Registration:** Utilizing Eureka for dynamic service discovery and registration, allowing microservices to find and communicate with each other seamlessly.
- **API Gateway**: Implementing Spring Cloud Gateway to provide a unified entry point for all client requests, routing them to the appropriate microservices.
- **Book Microservice:** Managing book-related operations, such as fetching book details and updating book availability.
- **Subscription Microservice:** Handling subscription-related operations, including creating new subscriptions and fetching subscription records.

This architecture ensures that the library system is robust, scalable, and easy to maintain, providing a reliable solution for managing library operations.

Technologies

The following technologies are used in the Book Library API:

- **Spring Boot**: A framework for building production-ready applications quickly and easily.
- **Spring Cloud Gateway**: Provides a simple, yet effective way to route requests to various microservices.
- **Eureka**: A service discovery tool from Netflix that allows microservices to register themselves and discover other services.
- **Spring Data JPA**: Simplifies data access and manipulation using Java Persistence API (JPA).
- **RestTemplate**: A synchronous client to perform HTTP requests, exposing a simple, template method API.
- **Swagger**: Provides interactive API documentation, allowing users to explore and test API endpoints directly from a web interface.
- **MySQL**: A relational database management system used to store and manage the library's data.

BookService API Endpoints

The BookService microservice provides several endpoints to manage book-related operations. Here is an overview of the available endpoints:

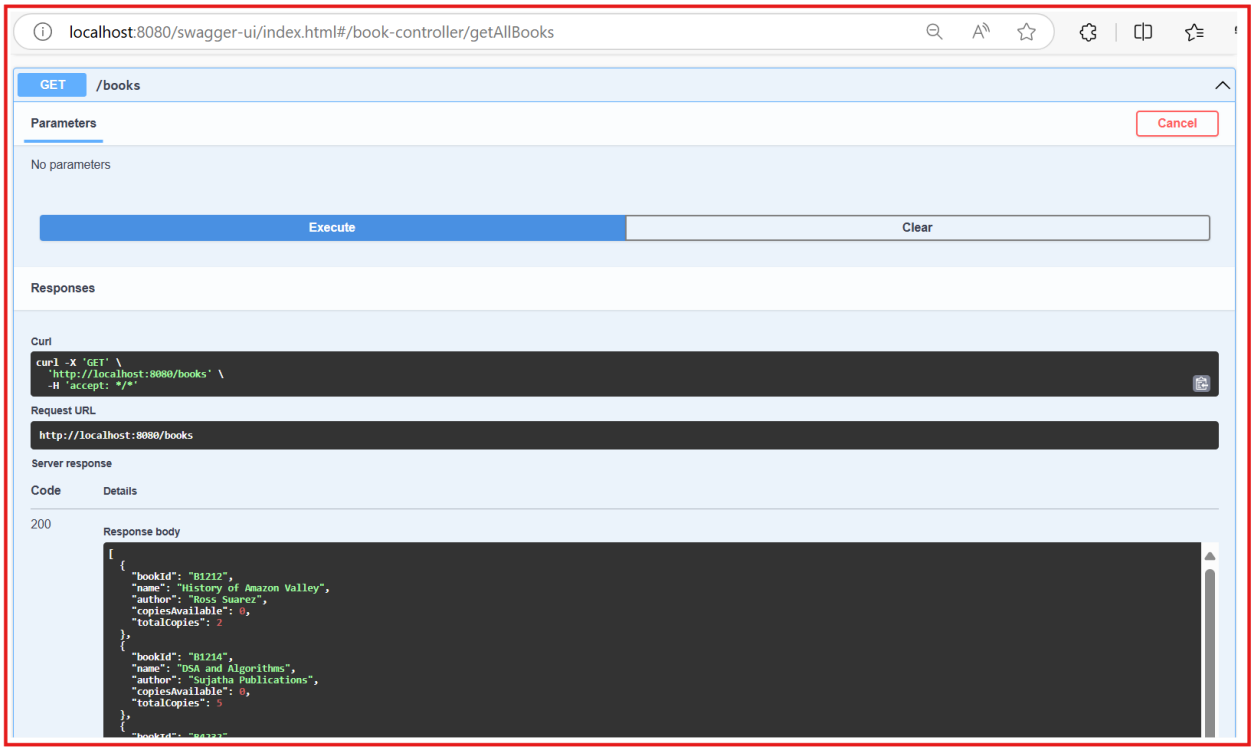
1.Get All Books

URL: GET /books

Description: Fetches a list of all books available in the library.

Response: A JSON array of book objects, each containing details such as book ID, name, author, available copies, and total copies.

HTTP Status Code: 200 OK



2.Get Book by ID

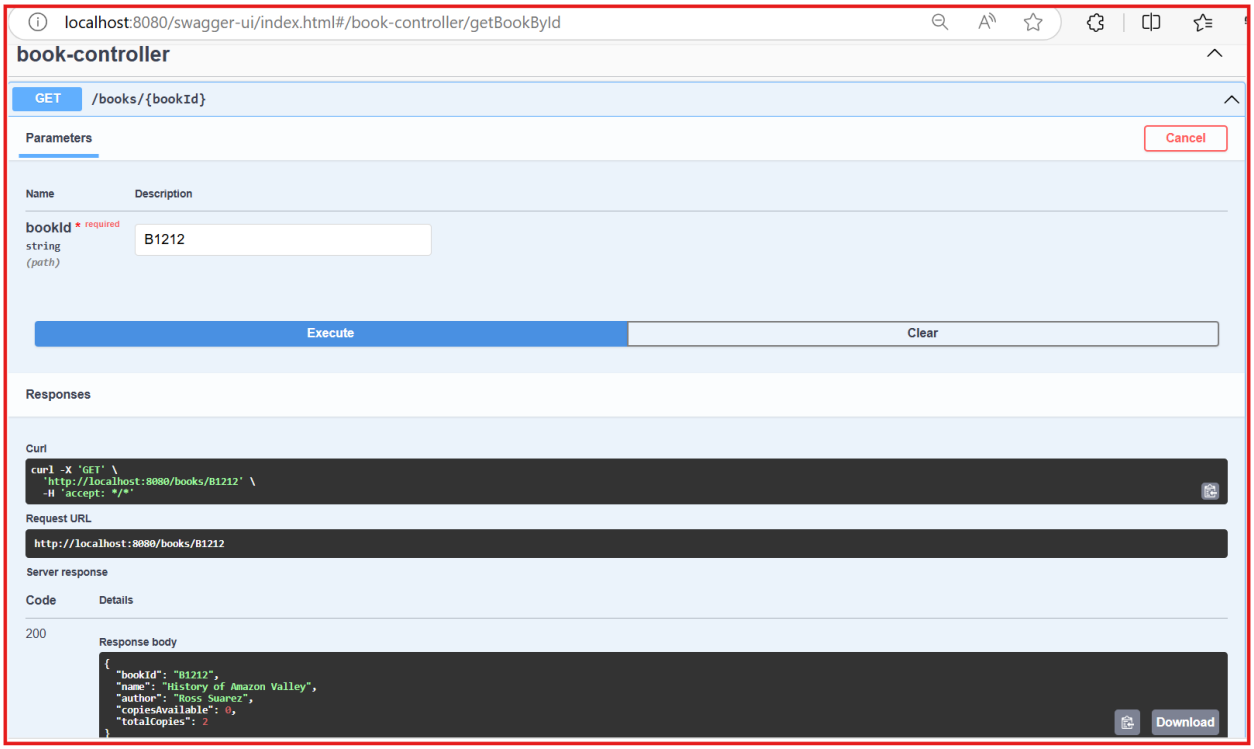
URL: GET /books/{bookId}

Description: Fetches details of a specific book by its ID.

Path Variable: bookId (String) - The ID of the book to fetch.

Response: A JSON object containing the book details if found.

HTTP Status Code: 200 OK if the book is found, 404 Not Found if the book does not exist.



3.Add a New Book

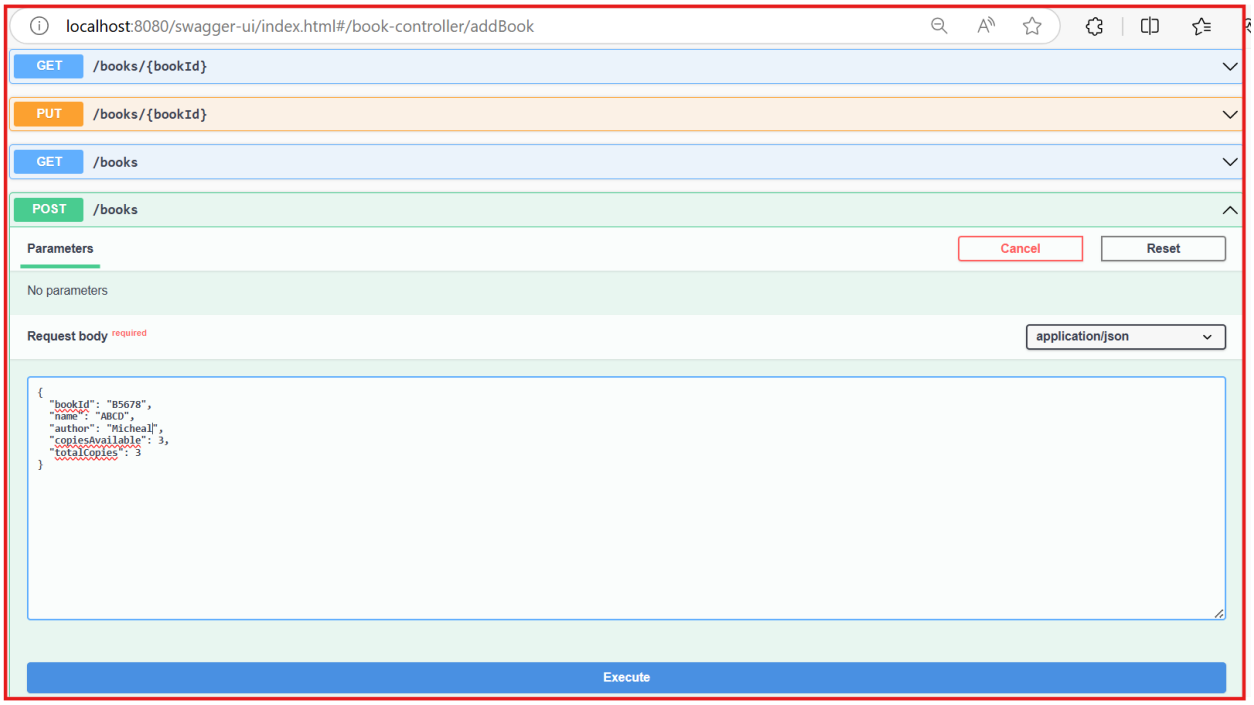
URL: POST /books

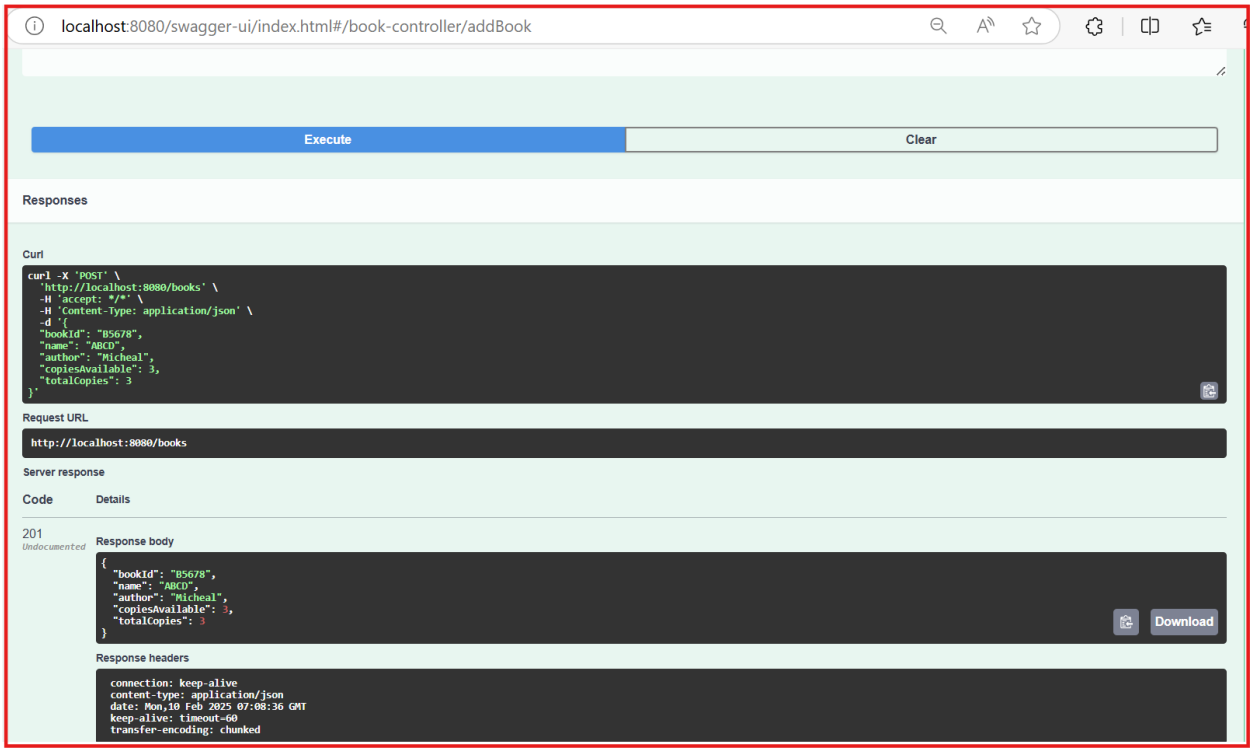
Description: Adds a new book to the library's inventory.

Request Body: A JSON object containing the details of the new book (book ID, name, author, available copies, total copies).

Response: A JSON object containing the saved book details.

HTTP Status Code: 201 Created





4.Update an Existing Book

URL: PUT /books/{bookId}

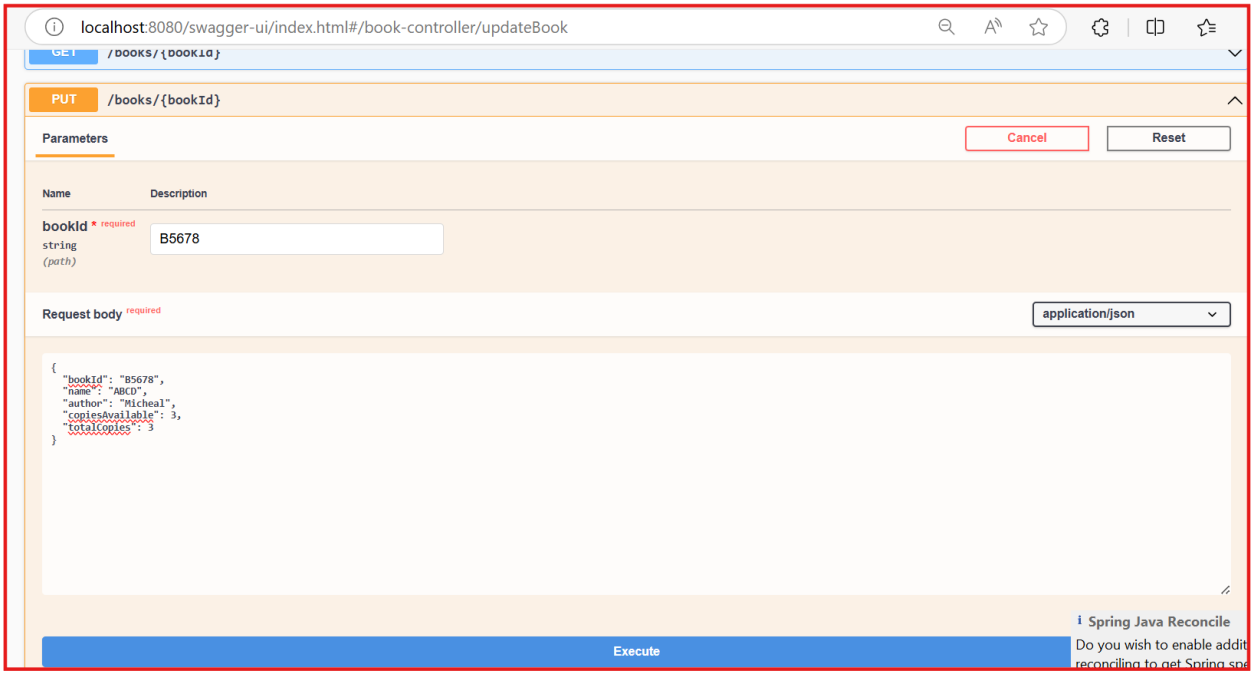
Description: Updates the details of an existing book by its ID.

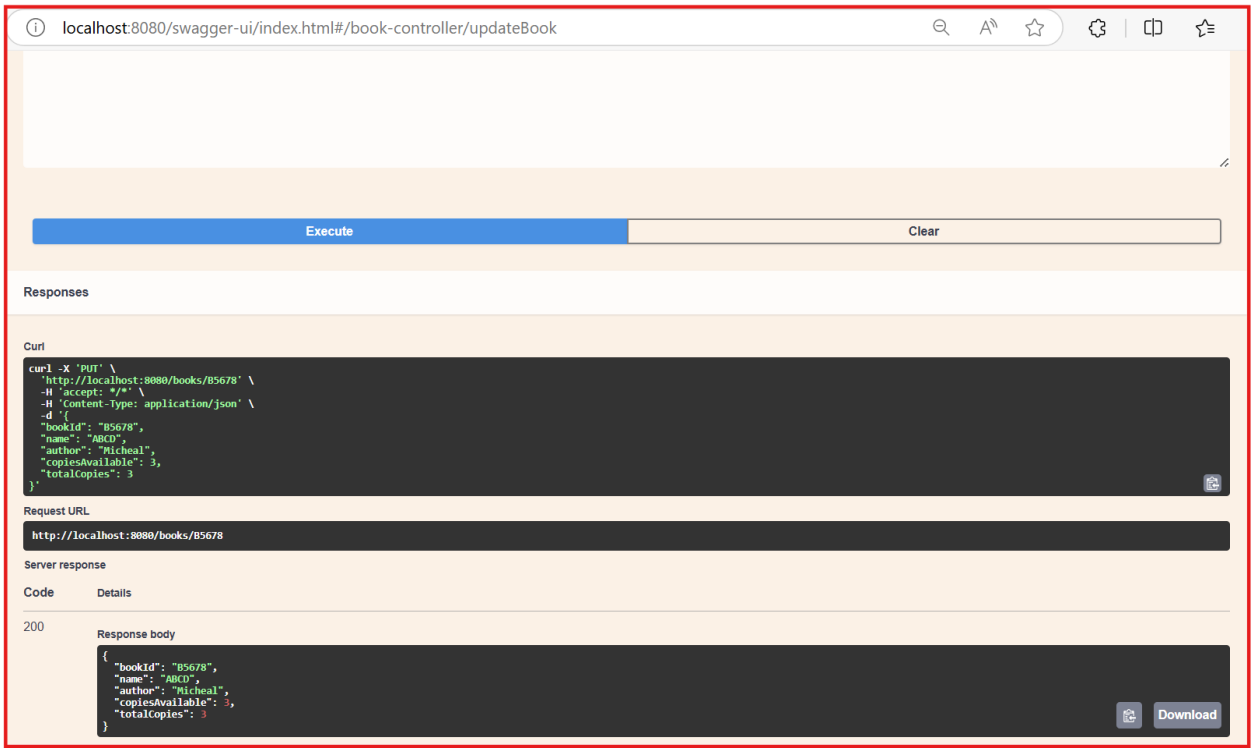
Path Variable: bookId (String) - The ID of the book to update.

Request Body: A JSON object containing the updated book details (name, author, available copies, total copies).

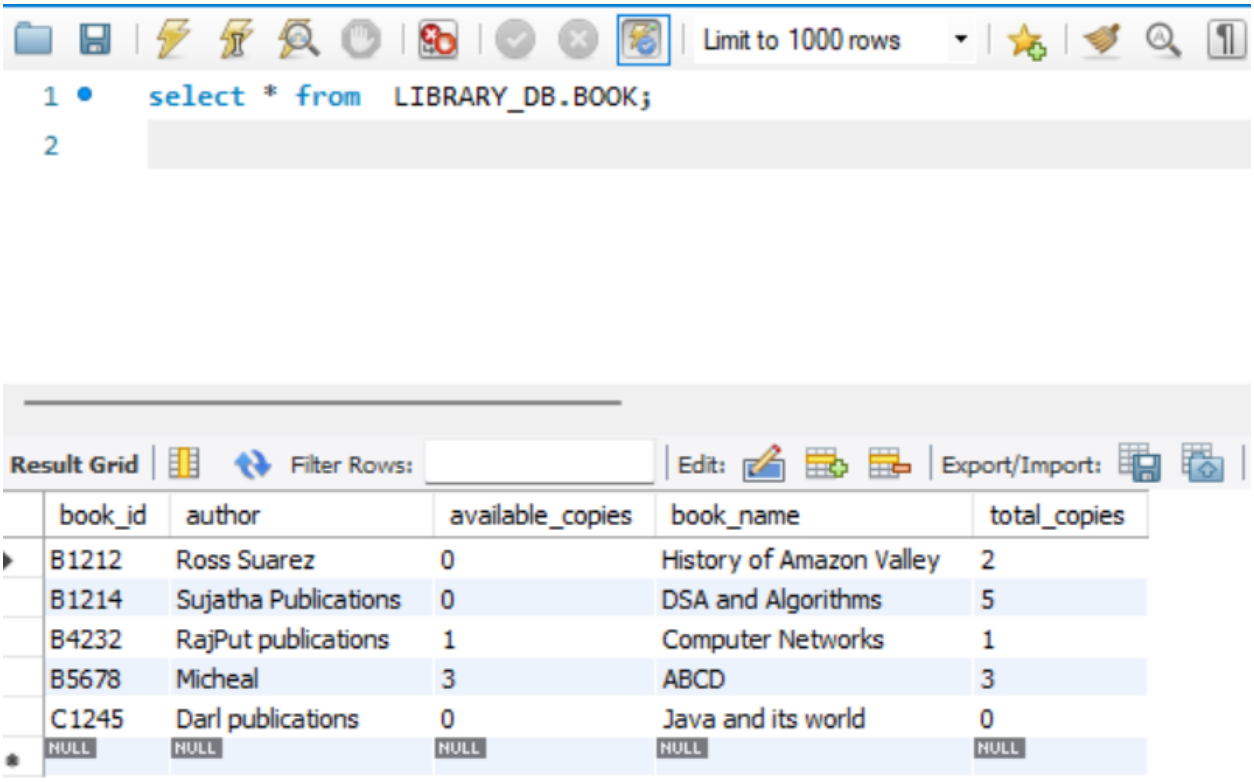
Response: A JSON object containing the updated book details if the update is successful.

HTTP Status Code: 200 OK if the update is successful, 404 Not Found if the book does not exist.





MYSQL DB: LIBRARY_DB.BOOK



book_id	author	available_copies	book_name	total_copies
B1212	Ross Suarez	0	History of Amazon Valley	2
B1214	Sujatha Publications	0	DSA and Algorithms	5
B4232	RajPut publications	1	Computer Networks	1
B5678	Micheal	3	ABCD	3
C1245	Darl publications	0	Java and its world	0
NULL	NULL	NULL	NULL	NULL

The database package for the BookService microservice includes components that interact with the database to perform CRUD operations on the BOOK table. Here are the key components:

1. **Entity Class (Book)**
   - Represents the BOOK table in the database.
   - Contains fields such as bookId, name, author, copiesAvailable, and totalCopies.
2. **Repository Interface (BookRepository)**
   - Extends Spring Data JPA's JpaRepository.
   - Provides methods for performing CRUD operations on the BOOK table.
3. **Service Class (BookService)**
   - Contains business logic for managing books.
   - Interacts with the repository to fetch and save data.
4. **Controller Class (BookController)**
   - Handles HTTP requests for book-related operations.
   - Defines endpoints for fetching all books, fetching a book by ID, adding a new book, and updating an existing book.

Swagger Integration

Swagger is integrated into the project to provide interactive API documentation. It allows users to explore and test API endpoints directly from a web interface.

- Swagger UI can be accessed at /swagger-ui.html. http://localhost:8080/swagger-ui/index.html

- Provides detailed information about each endpoint, including request parameters, response formats, and HTTP status codes.

SubscriptionService API Endpoints

The SubscriptionService microservice provides several endpoints to manage subscription-related operations. Here is an overview of the available endpoints:

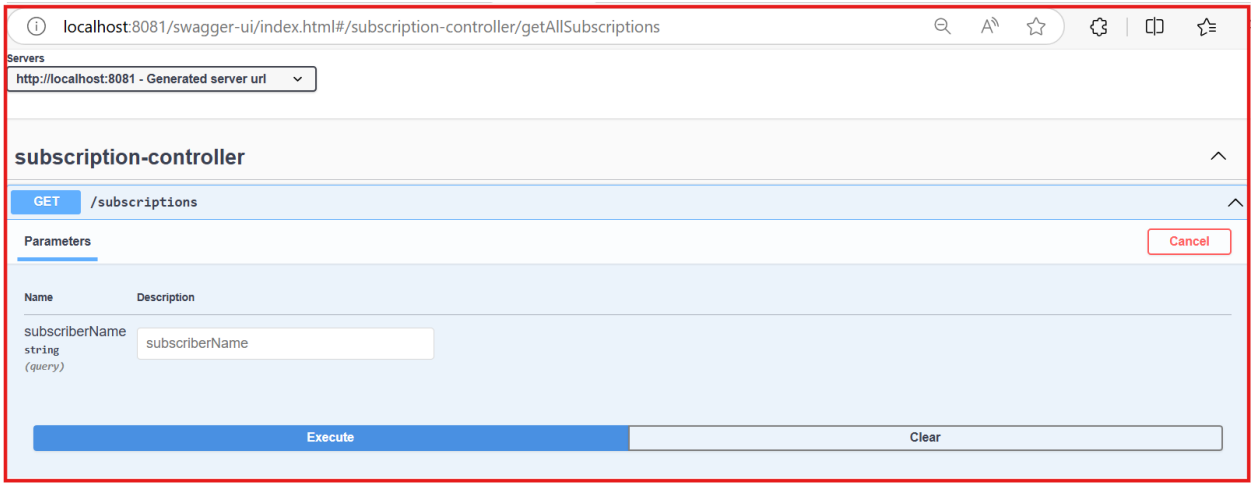
1.Get All Subscriptions

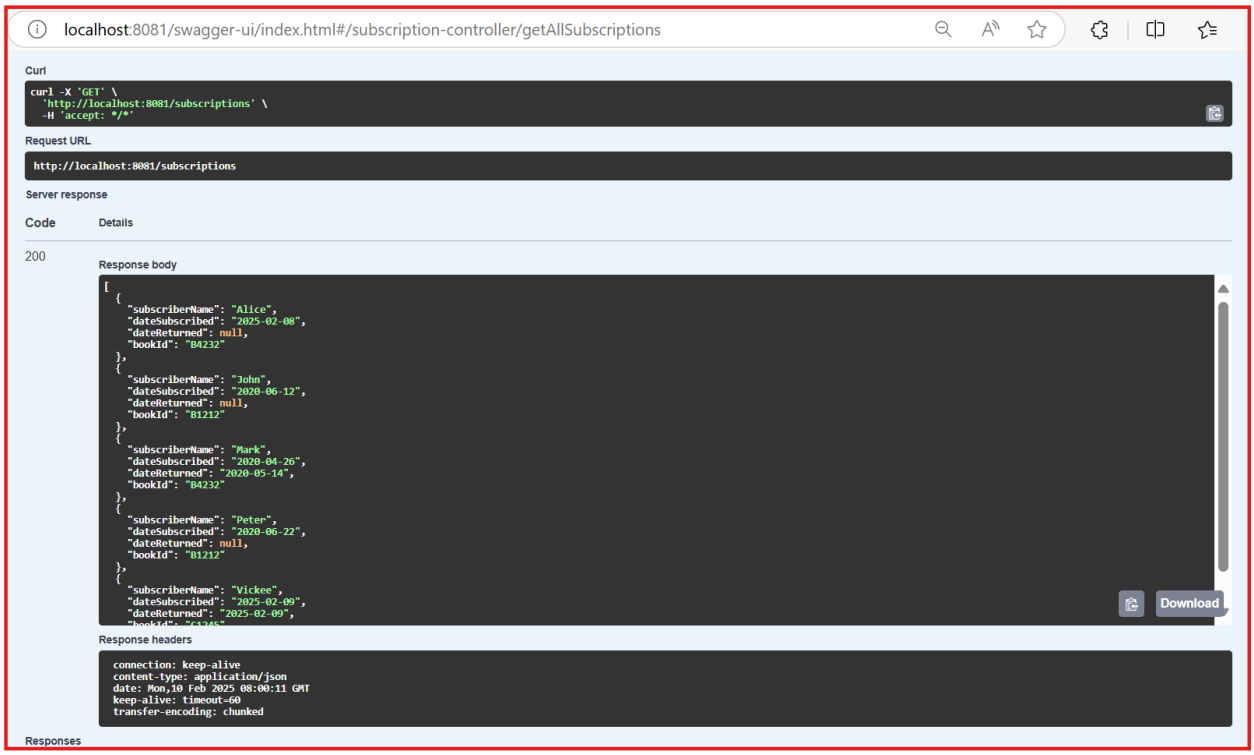
URL: GET /subscriptions

Description: Fetches a list of all subscriptions in the library.

Response: A JSON array of subscription objects, each containing details such as subscriber name, book ID, date subscribed, and date returned.

HTTP Status Code: 200 OK





2.Get Subscriptions by Subscriber Name

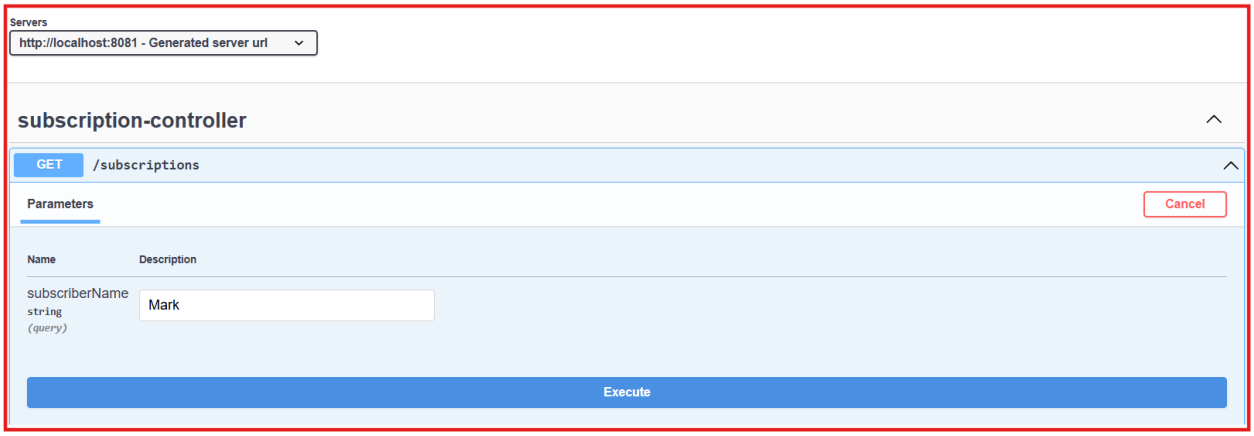
URL: GET /subscriptions?subscriberName={subscriberName}

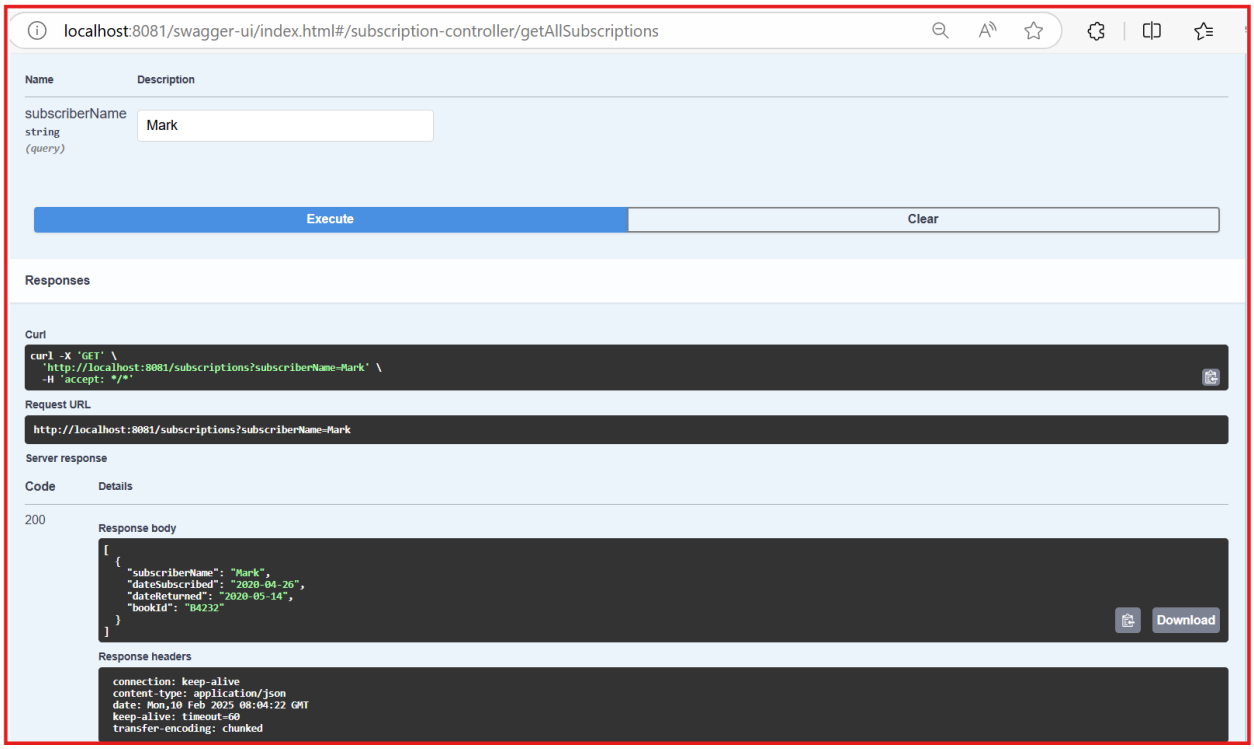
Description: Fetches a list of subscriptions for a specific subscriber.

Query Parameter: subscriberName (String) - The name of the subscriber to fetch subscriptions for.

Response: A JSON array of subscription objects for the specified subscriber.

HTTP Status Code: 200 OK





3.Create a New Subscription

URL: POST /subscriptions

Description: Creates a new subscription record.

Request Body: A JSON object containing the details of the new subscription (subscriber name, book ID, date subscribed).

Response: A JSON object containing the saved subscription details.

HTTP Status Code: 201 Created if the subscription is successfully created, 422 Unprocessable Entity if book copies are not available for subscription.

localhost:8081/swagger-ui/index.html#/subscription-controller/createSubscription

/

Example Value | Schema

```
string
```

POST /subscriptions

Parameters

Cancel Reset

No parameters

Request body required

application/json

```
{
  "subscriberName": "Ramu",
  "dateSubscribed": "2025-02-10",
  "dateReturned": "2025-02-10",
  "bookId": "B4232"
}
```

Execute Clear

localhost:8081/swagger-ui/index.html#/subscription-controller/createSubscription

Execute Clear

Responses

Curl

```
curl -X 'POST' \
  'http://localhost:8081/subscriptions' \
  -H 'accept: */*' \
  -H 'Content-Type: application/json' \
  -d '{
  "subscriberName": "Ramu",
  "dateSubscribed": "2025-02-10",
  "dateReturned": "2025-02-10",
  "bookId": "B4232"
}'
```

Request URL

```
http://localhost:8081/subscriptions
```

Server response

Code	Details
201	Response body

```
{
  "subscriberName": "Ramu",
  "dateSubscribed": "2025-02-10",
  "dateReturned": "2025-02-10",
  "bookId": "B4232"
}
```

Download

Response headers

```
connection: keep-alive
content-type: application/json
date: Mon,10 Feb 2025 08:10:35 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

MYSQL DB: LIBRARY_DB.SUBSCRIPTIONS

The database package for the SubscriptionService microservice includes components that interact with the database to perform CRUD operations on the SUBSCRIPTION table. Here are the key components:

```
2 •   SELECT * FROM LIBRARY_DB.SUBSCRIPTION;
```

Result Grid | Filter Rows: | Edit: | Exp

subscriber_name	book_id	date_returned	date_subscribed
Alice	B4232	NULL	2025-02-08
John	B1212	NULL	2020-06-12
Mark	B4232	2020-05-14	2020-04-26
Peter	B1212	NULL	2020-06-22
Ramu	B4232	2025-02-10	2025-02-10
Vickee	C1245	2025-02-09	2025-02-09
NULL	NULL	NULL	NULL

1.Entity Class (Subscription)

- Represents the SUBSCRIPTION table in the database.
- Contains fields such as id, subscriberName, bookId, dateSubscribed, and dateReturned.

2. **Repository Interface (SubscriptionRepository)**
   - Extends Spring Data JPA's JpaRepository.
   - Provides methods for performing CRUD operations on the SUBSCRIPTION table.
   - Includes a custom method to find subscriptions by subscriber name.
3. **Service Class (SubscriptionService)**
   - Contains business logic for managing subscriptions.
   - Interacts with the repository to fetch and save data.
   - Uses RestTemplate to call the book-service to check book availability before creating a subscription.
   - Handles transactional operations to ensure data consistency.
4. **Controller Class (SubscriptionController)**
   - Handles HTTP requests for subscription-related operations.

- Defines endpoints for fetching all subscriptions, fetching subscriptions by subscriber name, and creating a new subscription.

Swagger Integration

Swagger is integrated into the project to provide interactive API documentation. It allows users to explore and test API endpoints directly from a web interface.

- Swagger UI can be accessed at /swagger-ui.html. Swagger UI
- Provides detailed information about each endpoint, including request parameters, response formats, and HTTP status codes.

This documentation provides an overview of the key components and their roles in managing subscription-related operations within the Book Library API.

3. api-gateway-service

Configuration

The configuration for the api-gateway-service includes setting up the application name, server port, and Eureka client details for service discovery and registration. Additionally, it defines routes for the book-service and subscription-service to ensure that requests are correctly routed to these services.

Configuration Details:

- **Application Name**: The name of the application is set to api-gateway-service.
- **Server Port**: The server port is configured to 8082.
- **Eureka Client**: The Eureka client is configured to register with the Eureka server at http://localhost:8761/eureka/.
- **Logging Levels**: Debug logging is enabled for com.netflix.discovery and com.netflix.eureka to provide detailed logs for service discovery and registration.

Route Definitions:

1. **Books Route**
    - **Route ID**: books
    - **URI**: lb://books
    - **Predicate**: Routes requests with the path /books/** to the book-service.
2. **Subscriptions Route**
    - **Route ID**: subscriptions
    - **URI**: lb://subscriptions

- **Predicate**: Routes requests with the path /subscriptions/** to the subscription-service.

Configuration Example:

spring.application.name=api-gateway-service

server.port=8082

eureka.client.service-url.defaultZone=http://localhost:8761/eureka/

logging.level.com.netflix.discovery=DEBUG

logging.level.com.netflix.eureka=DEBUG

spring.cloud.gateway.routes[0].id=books

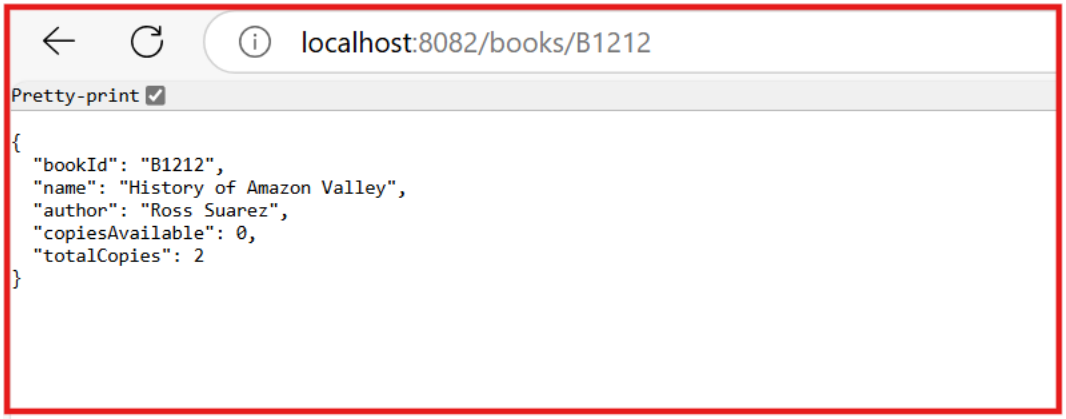
spring.cloud.gateway.routes[0].uri=lb://books

spring.cloud.gateway.routes[0].predicates[0]=Path=/books/**

spring.cloud.gateway.routes[1].id=subscriptions

spring.cloud.gateway.routes[1].uri=lb://subscriptions

spring.cloud.gateway.routes[1].predicates[0]=Path=/subscriptions/**



localhost:8082/subscriptions?subscriberName=Mark

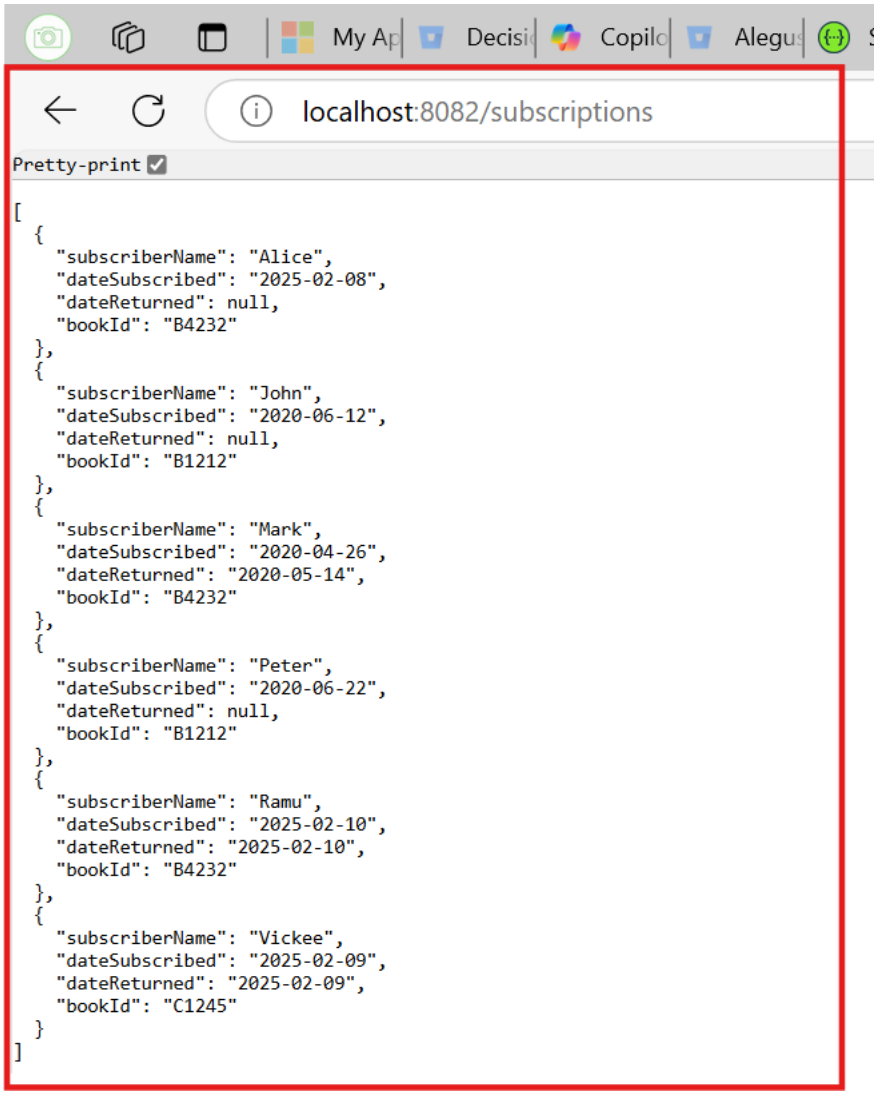
Pretty-print


```
[
  {
    "subscriberName": "Mark",
    "dateSubscribed": "2020-04-26",
    "dateReturned": "2020-05-14",
    "bookId": "B4232"
  }
]
```


localhost:8082/books

Pretty-print


```
[
  {
    "bookId": "B1212",
    "name": "History of Amazon Valley",
    "author": "Ross Suarez",
    "copiesAvailable": 0,
    "totalCopies": 2
  },
  {
    "bookId": "B1214",
    "name": "DSA and Algorithms",
    "author": "Sujatha Publications",
    "copiesAvailable": 0,
    "totalCopies": 5
  },
  {
    "bookId": "B4232",
    "name": null,
    "author": null,
    "copiesAvailable": 0,
    "totalCopies": 0
  },
  {
    "bookId": "B5678",
    "name": "ABCD",
    "author": "Micheal",
    "copiesAvailable": 3,
    "totalCopies": 3
  },
  {
    "bookId": "C1245",
    "name": "Java and its world",
    "author": "Darl publications",
    "copiesAvailable": 0,
    "totalCopies": 0
  }
]
```




```
[
  {
    "subscriberName": "Alice",
    "dateSubscribed": "2025-02-08",
    "dateReturned": null,
    "bookId": "B4232"
  },
  {
    "subscriberName": "John",
    "dateSubscribed": "2020-06-12",
    "dateReturned": null,
    "bookId": "B1212"
  },
  {
    "subscriberName": "Mark",
    "dateSubscribed": "2020-04-26",
    "dateReturned": "2020-05-14",
    "bookId": "B4232"
  },
  {
    "subscriberName": "Peter",
    "dateSubscribed": "2020-06-22",
    "dateReturned": null,
    "bookId": "B1212"
  },
  {
    "subscriberName": "Ramu",
    "dateSubscribed": "2025-02-10",
    "dateReturned": "2025-02-10",
    "bookId": "B4232"
  },
  {
    "subscriberName": "Vickee",
    "dateSubscribed": "2025-02-09",
    "dateReturned": "2025-02-09",
    "bookId": "C1245"
  }
]
```

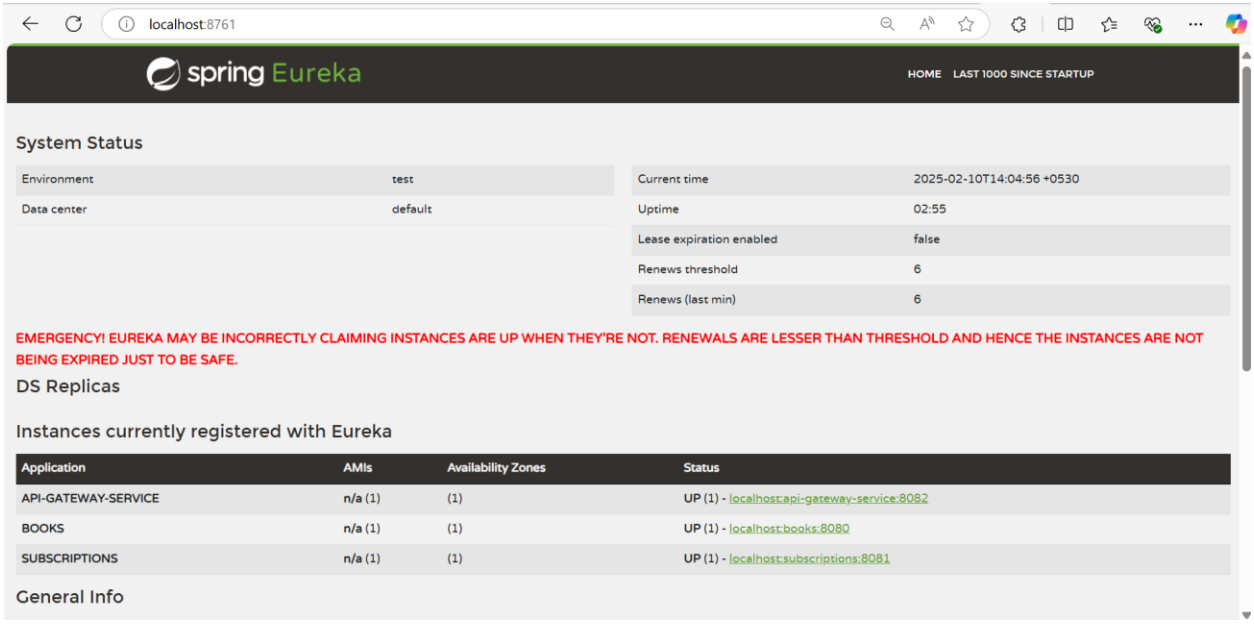

4. service-discovery

- **Eureka Server**: This service uses Eureka for service discovery and registration. All other microservices register themselves with the Eureka server, which allows them to discover each other and communicate.

Running the Application

1. **Start the Eureka server**: This service should be started first to allow other services to register themselves.
2. **Start the book-service**: This service manages book-related operations and should be started next.

3. **Start the subscription-service**: This service manages subscription-related operations and checks book availability using the book-service.
4. **Start the api-gateway-service**: This service routes requests to the appropriate microservices based on the URL patterns.



Conclusion

In conclusion, the Book Library API project successfully implements a microservices architecture to manage a library's book inventory and subscriber records. By creating two microservices, book-service and subscription-service, and integrating them with an API Gateway and Eureka for service discovery and registration, we have achieved a scalable and maintainable system.

The key components and their roles are as follows:

- **Book Microservice**: Manages book-related operations, including fetching book details and updating book availability.
- **Subscription Microservice**: Handles subscription-related operations, such as creating new subscriptions and fetching subscription records. It also interacts with the Book Microservice to check book availability before creating a subscription.
- **API Gateway**: Acts as a single entry point for all client requests, routing them to the appropriate microservices based on URL patterns.
- **Service Discovery and Registration**: Utilizes Eureka to enable dynamic service discovery and registration, allowing microservices to find and communicate with each other seamlessly.

By following the standard pattern of REST Controller -> Service -> Repository, and using technologies such as Spring Boot, Spring Cloud Gateway, Eureka, Spring Data JPA, RestTemplate, Swagger, and MySQL, we have created a robust and efficient library management system.