BookStore API Overview

XIDERAL
EVER HINOJOSA AGUIRRE

Description

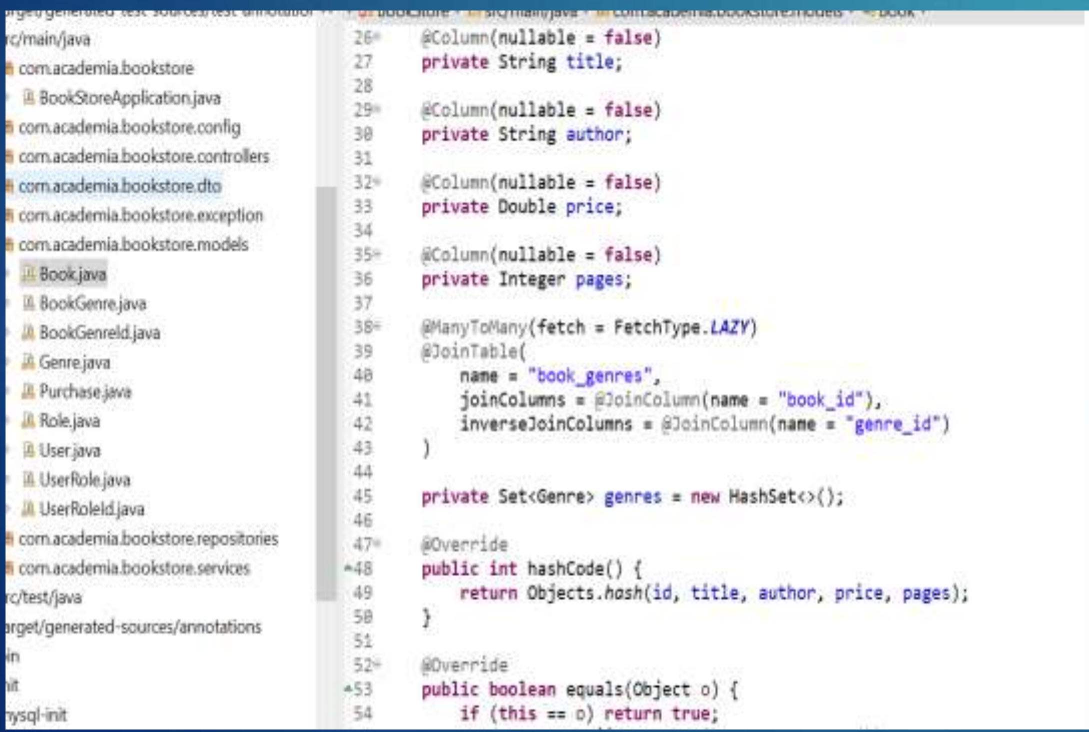
- A Restful api to manage bookstore operations. It features functionality for managing books, genres, users, and roles, with robust support for searching, filtering, and purchasing books

Technology Stack

- **Java**: 17
- **Spring Boot**: 3.3.3
- **Spring Data JPA**: Integrated with Spring Boot version
- **Spring Security**: Integrated with Spring Boot version
- **MySQL**: 8.0.39
- **RabbitMQ**: Used for message brokering.
- **Docker**: Used for containerization (24.0.6).
- **Docker Compose**: Used for multi-container orchestration.
- **Maven**: Used for project management and build automation

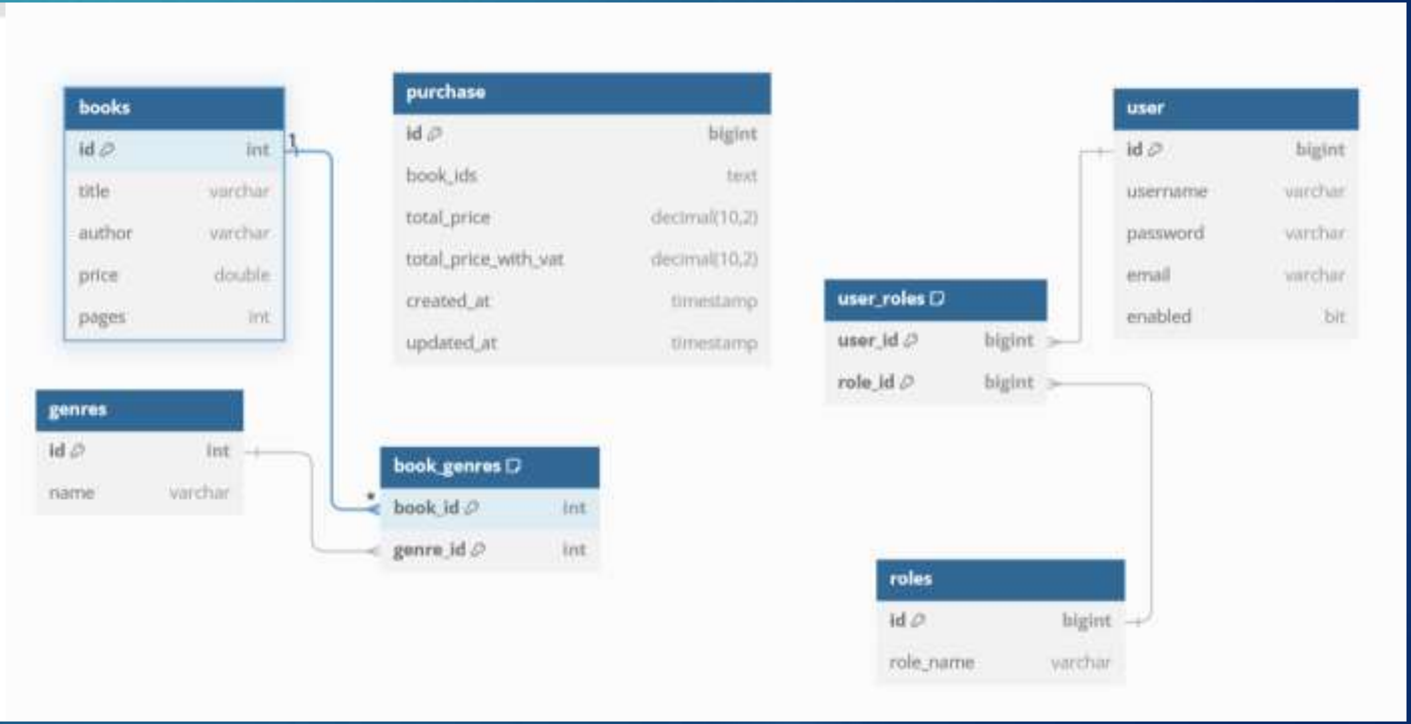
Database and Data JPA

- Has a Docker container, initialized with 3 .sql in the mysql-init directory.



```
rc/main/java
  com.academia.bookstore
    BookStoreApplication.java
  com.academia.bookstore.config
  com.academia.bookstore.controllers
  com.academia.bookstore.dto
  com.academia.bookstore.exception
  com.academia.bookstore.models
    Book.java
    BookGenre.java
    BookGenreId.java
    Genre.java
    Purchase.java
    Role.java
    User.java
    UserRole.java
    UserRoleId.java
  com.academia.bookstore.repositories
  com.academia.bookstore.services
rc/test/java
arget/generated-sources/annotations
in
it
ysql-init
```

```java
    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String author;

    @Column(nullable = false)
    private Double price;

    @Column(nullable = false)
    private Integer pages;

    @ManyToMany(fetch = FetchType.LAZY)
    @JoinTable(
        name = "book_genres",
        joinColumns = @JoinColumn(name = "book_id"),
        inverseJoinColumns = @JoinColumn(name = "genre_id")
    )

    private Set<Genre> genres = new HashSet<>();

    @Override
    public int hashCode() {
        return Objects.hash(id, title, author, price, pages);
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
```



books	
id	int
title	varchar
author	varchar
price	double
pages	int

purchase	
id	bigint
book_ids	text
total_price	decimal(10,2)
total_price_with_vat	decimal(10,2)
created_at	timestamp
updated_at	timestamp

user	
id	bigint
username	varchar
password	varchar
email	varchar
enabled	bit

user_roles	
user_id	bigint
role_id	bigint

genres	
id	int
name	varchar

book_genres	
book_id	int
genre_id	int

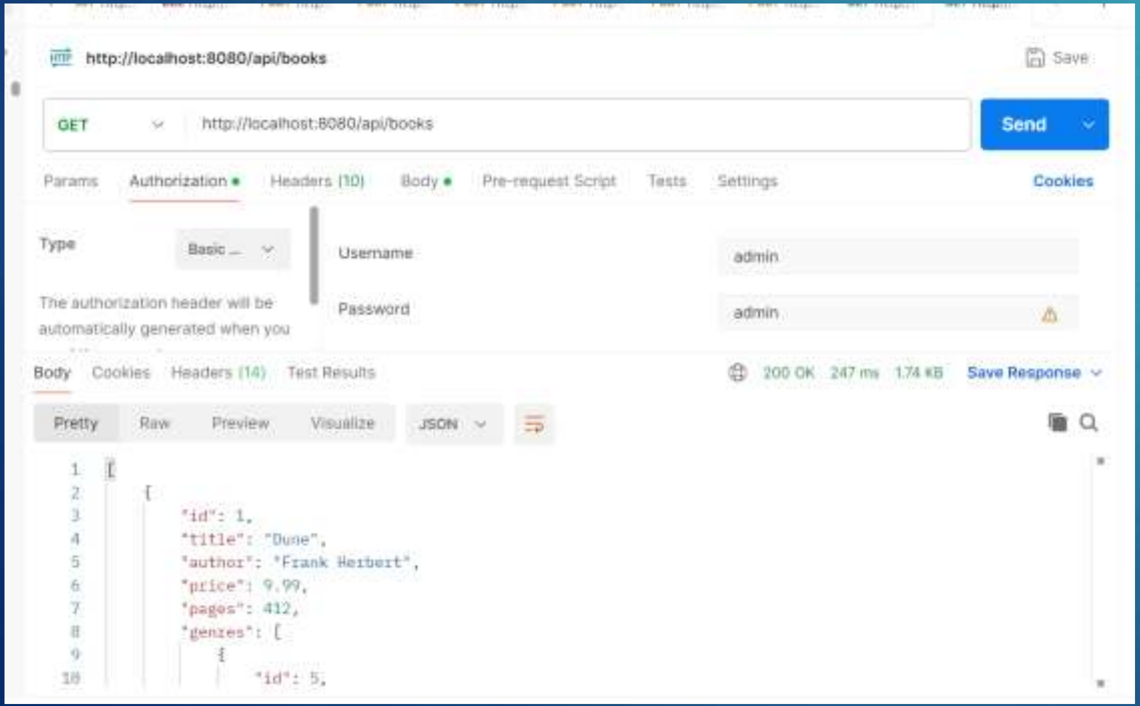
roles	
id	bigint
role_name	varchar

Endpoint Security Tests

- Books List:

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

- admin role will have Access to all the endpoints in the api.
- User role will have Access only to store endpoints.



http://localhost:8080/api/books

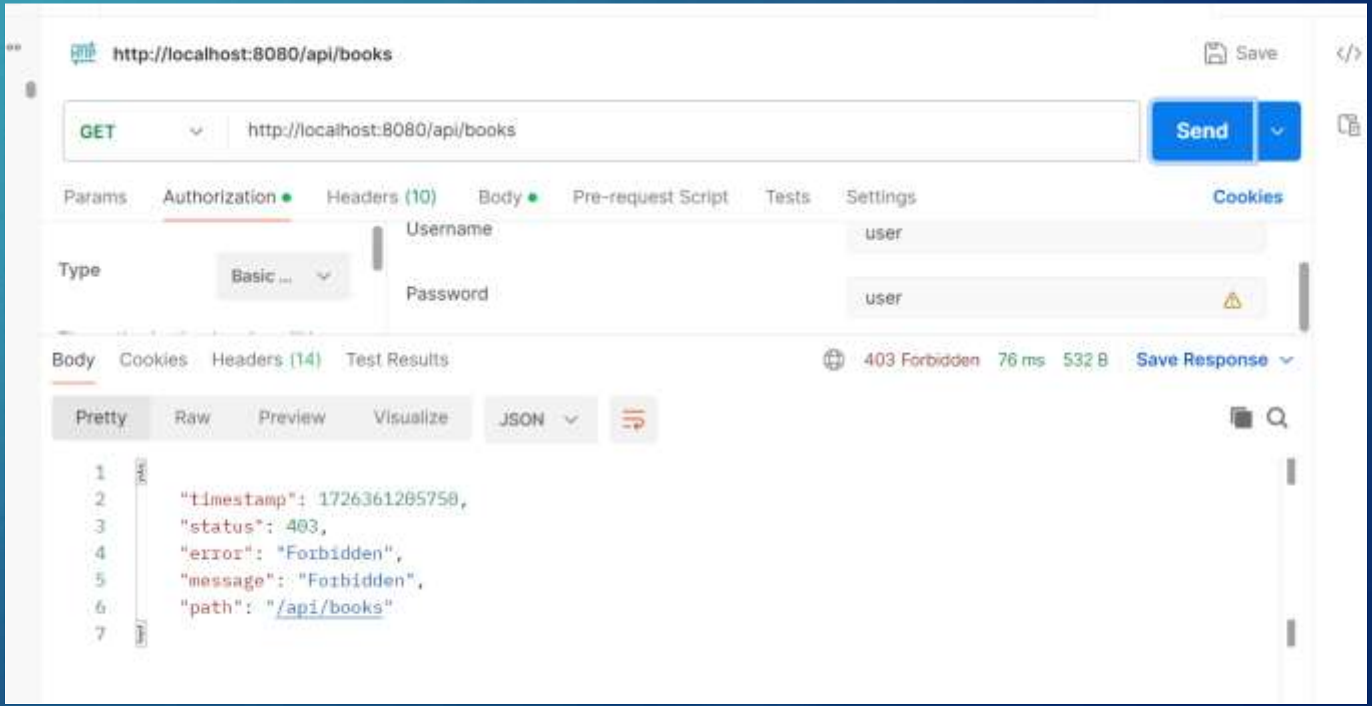
GET http://localhost:8080/api/books

Authorization: Type Basic, Username admin, Password admin

The authorization header will be automatically generated when you

200 OK 247 ms 1.74 KB

```
[
    {
        "id": 1,
        "title": "Dune",
        "author": "Frank Herbert",
        "price": 9.99,
        "pages": 412,
        "genres": [
            {
                "id": 5,
```



http://localhost:8080/api/books

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

Authorization: Type Basic, Username user, Password user

403 Forbidden 76 ms 532 B

```
{
    "timestamp": 1726361205750,
    "status": 403,
    "error": "Forbidden",
    "message": "Forbidden",
    "path": "/api/books"
}
```

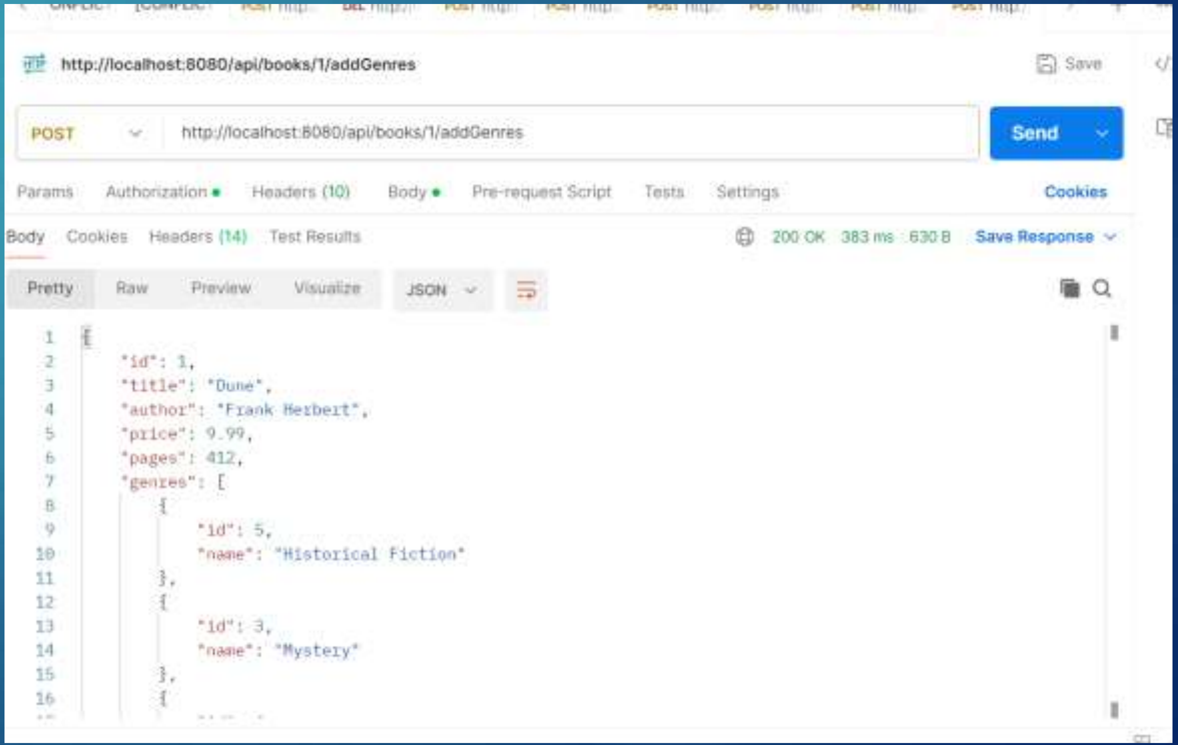
Endpoint tests

- Add multiple genres to a book
- POST(Bookid in param) http://localhost:8080/api/books/1/addGenres
- JSON(Genreids): [3, 4, 5]

Books, Genres, Users and Roles

All have their respective CRUDs and

Some extras


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

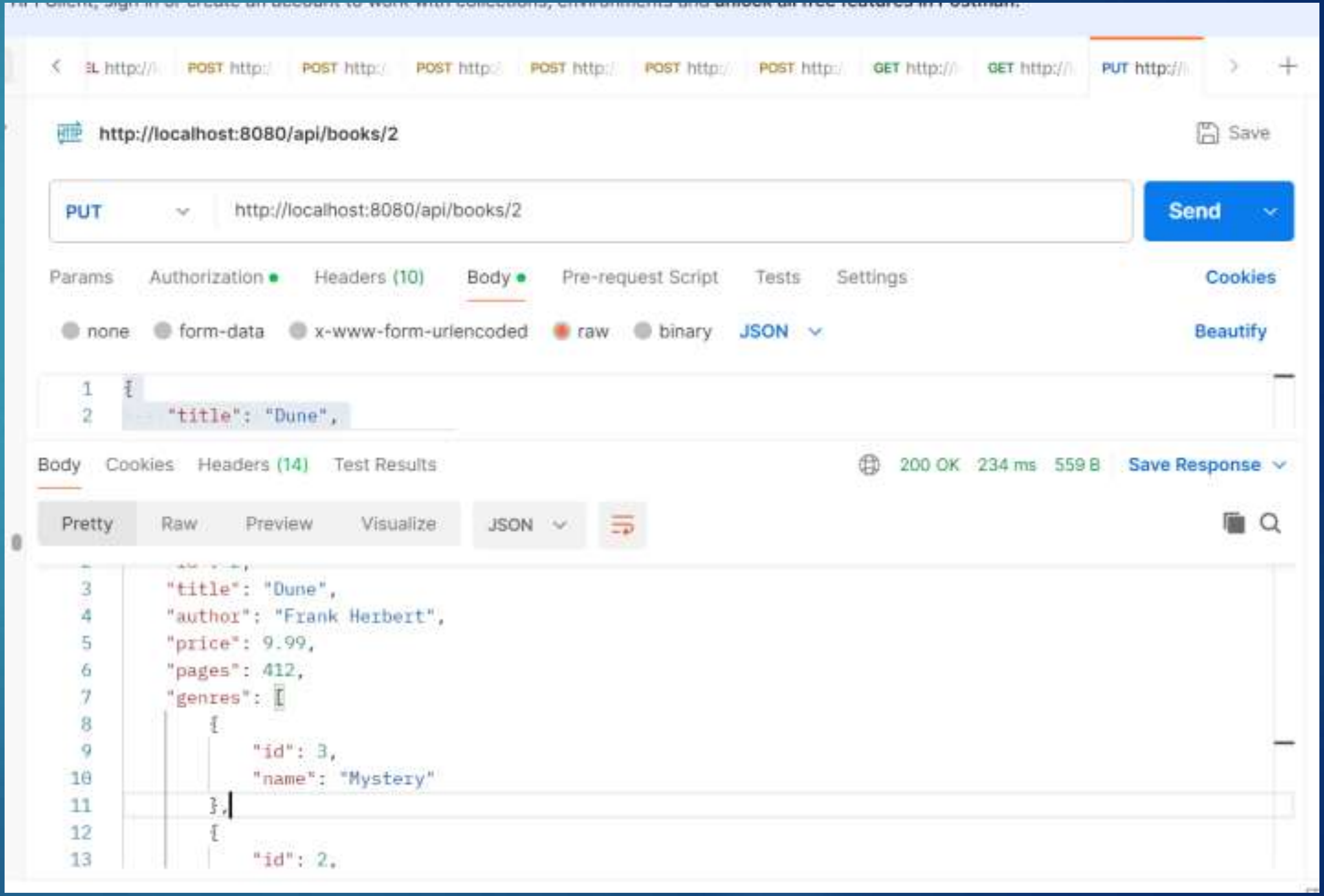
POST http://localhost:8080/api/books/1/addGenres

200 OK 383 ms 630 B

```
{
    "id": 1,
    "title": "Dune",
    "author": "Frank Herbert",
    "price": 9.99,
    "pages": 412,
    "genres": [
        {
            "id": 5,
            "name": "Historical Fiction"
        },
        {
            "id": 3,
            "name": "Mystery"
        },
        {
```

Endpoint tests

- Edit a Book
- PUT(Bookid in param) http://localhost:8080/api/books/2
- JSON(Book):
- {
- "title": "Dune",
- "author": "Frank Herbert",
- "price": 9.99,
- "pages": 412,
- "genres": [
- {
- "id": 2,
- "name": "Fantasy"
- },
- {
- "id": 3,
- "name": "Mystery"
- }
-]
- }



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

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

```
{
    "title": "Dune",
```

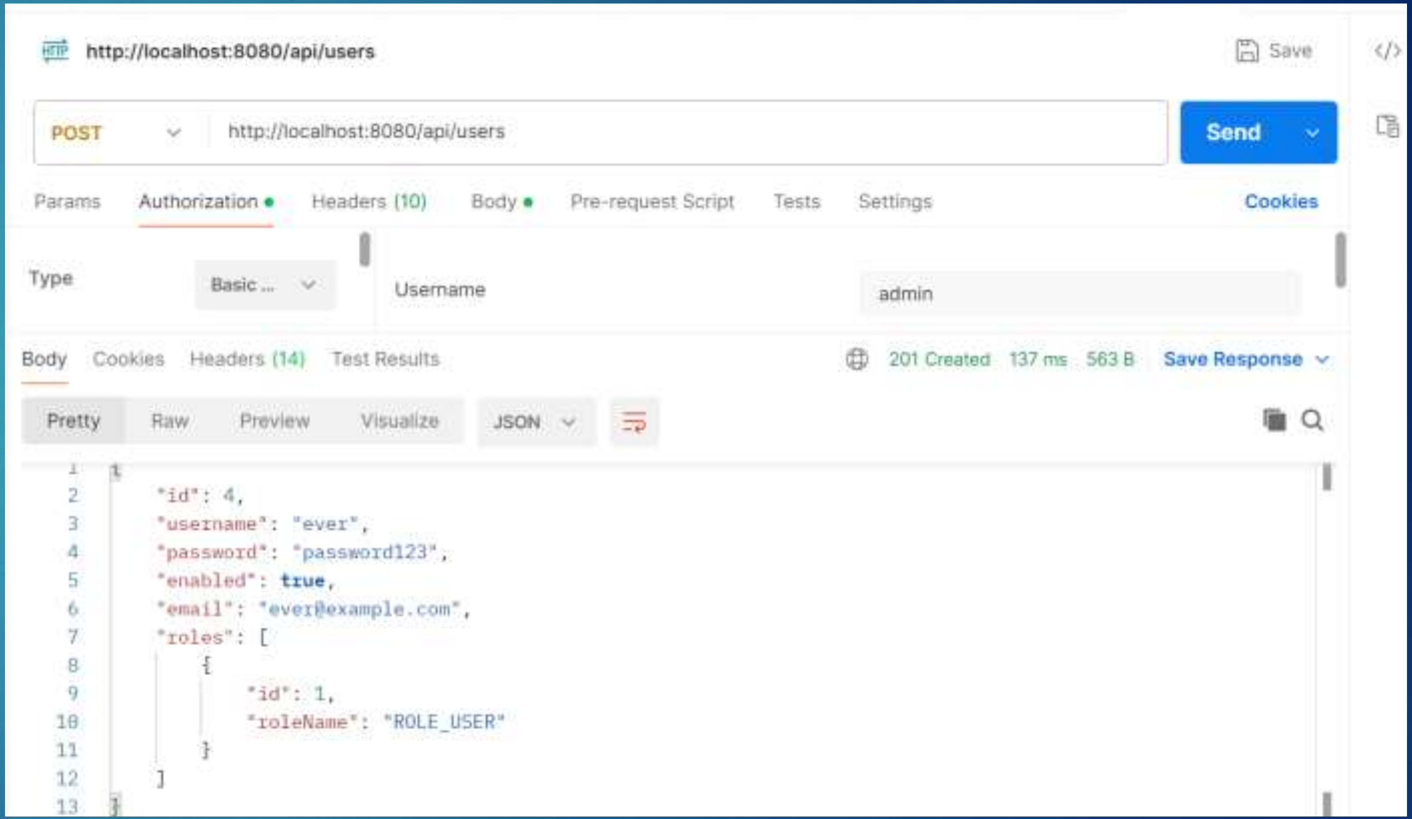
200 OK 234 ms 559 B

```
    "title": "Dune",
    "author": "Frank Herbert",
    "price": 9.99,
    "pages": 412,
    "genres": [
        {
            "id": 3,
            "name": "Mystery"
        },
        {
            "id": 2,
```

EL http:// POST http:// POST http:// POST http:// POST http:// POST http:// POST http:// GET http:// GET http:// PUT http://
Save
Send
Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies
none form-data x-www-form-urlencoded raw binary JSON Beautify
Body Cookies Headers (14) Test Results Save Response
Pretty Raw Preview Visualize JSON

Endpoint tests

- Create a User
- POST
- http://localhost:8080/api/users
- JSON:
- {
-     "username": "ever",
-     "password": "password123",
-     "email": "ever@example.com",
-     "roleNames": ["ROLE_USER"]
- }



http://localhost:8080/api/users

POST http://localhost:8080/api/users

Type: Basic ...

Username: admin

201 Created 137 ms 563 B

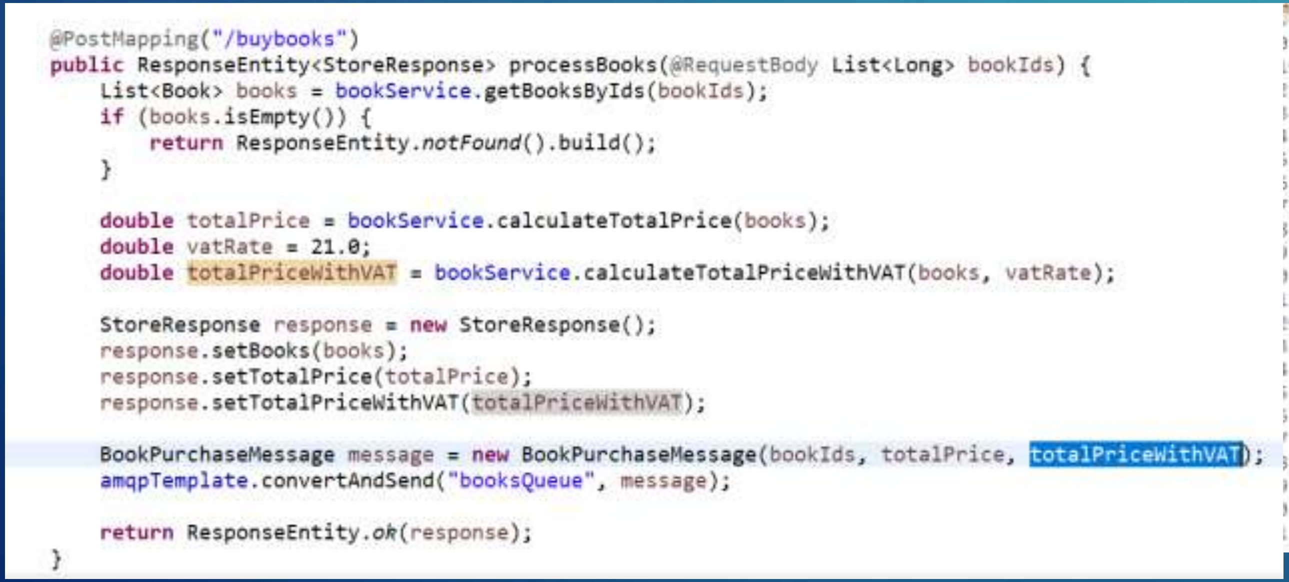
```
{
    "id": 4,
    "username": "ever",
    "password": "password123",
    "enabled": true,
    "email": "ever@example.com",
    "roles": [
        {
            "id": 1,
            "roleName": "ROLE_USER"
        }
    ]
}
```

RabbitMQ

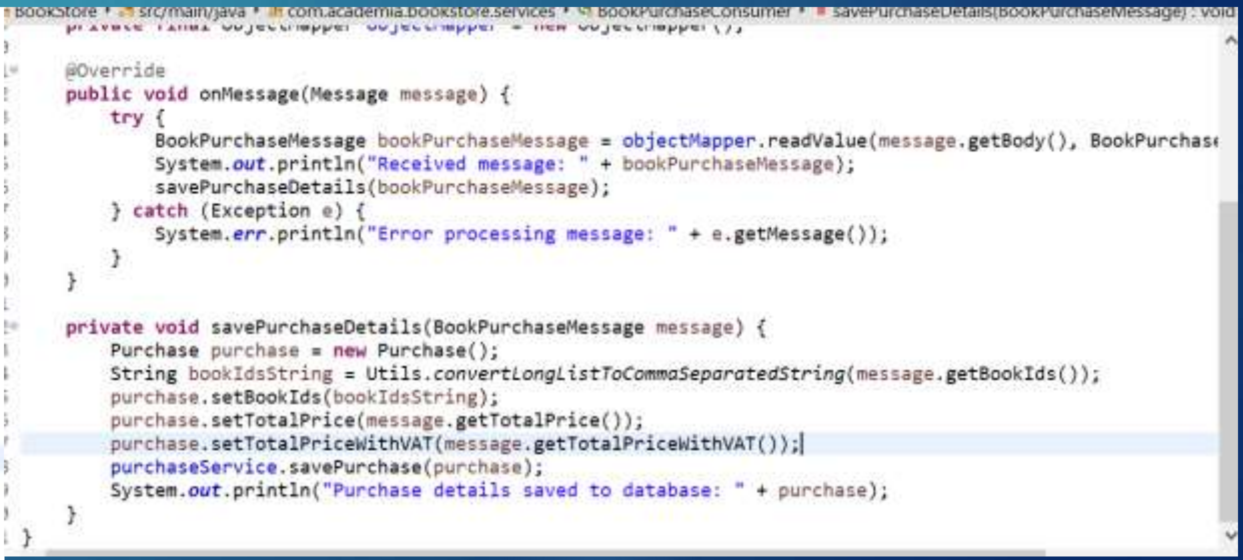
- Also has a Docker container

- The idea is to send a message indicating that books have been purchased and to save this information in the database.

- When using /buybooks a list of bookIds is sent and when received saved in the DB



```java
@PostMapping("/buybooks")
public ResponseEntity<StoreResponse> processBooks(@RequestBody List<Long> bookIds) {
    List<Book> books = bookService.getBooksByIds(bookIds);
    if (books.isEmpty()) {
        return ResponseEntity.notFound().build();
    }

    double totalPrice = bookService.calculateTotalPrice(books);
    double vatRate = 21.0;
    double totalPriceWithVAT = bookService.calculateTotalPriceWithVAT(books, vatRate);

    StoreResponse response = new StoreResponse();
    response.setBooks(books);
    response.setTotalPrice(totalPrice);
    response.setTotalPriceWithVAT(totalPriceWithVAT);

    BookPurchaseMessage message = new BookPurchaseMessage(bookIds, totalPrice, totalPriceWithVAT);
    amqpTemplate.convertAndSend("booksQueue", message);

    return ResponseEntity.ok(response);
}
```



BookStore > src/main/java > com.academia.bookstore.services > BookPurchaseConsumer > savePurchaseDetails(BookPurchaseMessage) : void

```java
    @Override
    public void onMessage(Message message) {
        try {
            BookPurchaseMessage bookPurchaseMessage = objectMapper.readValue(message.getBody(), BookPurchase
            System.out.println("Received message: " + bookPurchaseMessage);
            savePurchaseDetails(bookPurchaseMessage);
        } catch (Exception e) {
            System.err.println("Error processing message: " + e.getMessage());
        }
    }

    private void savePurchaseDetails(BookPurchaseMessage message) {
        Purchase purchase = new Purchase();
        String bookIdsString = Utils.convertLongListToCommaSeparatedString(message.getBookIds());
        purchase.setBookIds(bookIdsString);
        purchase.setTotalPrice(message.getTotalPrice());
        purchase.setTotalPriceWithVAT(message.getTotalPriceWithVAT());
        purchaseService.savePurchase(purchase);
        System.out.println("Purchase details saved to database: " + purchase);
    }
}
```

Thank you

Commands

Start up containers:

```
docker-compose -f compose.yaml -f compose-r.yaml up
```

Check up tables:

```
docker exec -it bookstore-mysql-1 mysql -u root -p
```