



BOOKSTORE API

Objective

This project demonstrates the application of key concepts and technologies learned during the Xideral Java bootcamp, showcasing the ability to build a full-featured RESTful API using modern Java technologies.

Description

Bookstore is a Java-based web application developed with Spring Boot to manage bookstore operations. It features functionality for managing books, genres, users, and roles, with robust support for searching, filtering, and purchasing books

Features:

Book Management - ROLE_ADMIN required

- **CRUD Operations:** Create, Read, Update, and Delete books with full support for associating books with genres.
- **Search and Filter:** Search books by title, retrieve books by genre, and filter books based on price.

Genre Management - ROLE_ADMIN required

- **CRUD Operations:** Manage genres by creating, updating, and deleting genre entries.
- **Book-Genre Association:** Link books with multiple genres and manage these associations efficiently.

Role Management - ROLE_ADMIN required

- **Role Management:** Create, retrieve, update, and delete user roles, which can be assigned to users to define their access levels.

User Management - ROLE_ADMIN required

- **User Operations:** Create, update, delete, and retrieve users with role-based access control.
- **Role Assignment:** Add or remove roles for users dynamically to manage user permissions effectively.

Store Operations - ROLE_USER required

- **Book Retrieval:** Retrieve all books or search based on title, genre, or price range.
- **Purchase Processing:** Handle book purchases with detailed responses including total price calculations and VAT.

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

ERD



Tables:

- **Books:** Represents book details including title, author, price, and pages.
- **Genres:** Represents different genres of books.
- **Users:** Represents users of the system with associated roles.
- **Roles:** Defines various user roles within the system.
- **Purchases:** Records transactions made by users.

Relationships:

- **Books to Genres:** Many-to-many relationship, managed via the book_genres table.
- **Users to Roles:** Many-to-many relationship, managed via the user_roles table.
- **Purchases:** Each purchase can include multiple books, but each purchase is associated with a single user.

Bookstore API - Controllers Overview

API Endpoints

Books

- **Get All Books**
 - **URL:** `GET /api/books`
 - **Description:** Retrieves a list of all books in the system.
 - **Response:** `200 OK` with a list of books.
- **Get Book by ID**
 - **URL:** `GET /api/books/{id}`
 - **Description:** Retrieves a book by its ID.
 - **Path Parameter:** `id` (Long) - The ID of the book.
 - **Response:** `200 OK` with the book details or `404 Not Found` if the book does not exist.
- **Create Book**
 - **URL:** `POST /api/books`
 - **Description:** Creates a new book.
 - **Request Body:** Book details.
 - **Response:** `201 Created` with the created book.
- **Add Genres to Book**
 - **URL:** `POST /api/books/{bookId}/addGenres`
 - **Description:** Adds genres to an existing book.
 - **Path Parameter:** `bookId` (Long) - The ID of the book.
 - **Request Body:** List of genre IDs.
 - **Response:** `200 OK` with the updated book or `404 Not Found` if the book or genre does not exist.

- **Update Book**
 - **URL:** PUT /api/books/{id}
 - **Description:** Updates an existing book.
 - **Path Parameter:** id (Long) - The ID of the book.
 - **Request Body:** Updated book details.
 - **Response:** 200 OK with the updated book or 404 Not Found if the book does not exist.

- **Delete Book**
 - **URL:** DELETE /api/books/{id}
 - **Description:** Deletes a book by its ID.
 - **Path Parameter:** id (Long) - The ID of the book.
 - **Response:** 204 No Content or 404 Not Found if the book does not exist.

- **Search Books by Title**
 - **URL:** GET /api/books/search
 - **Description:** Searches for books by title.
 - **Query Parameter:** title (String) - The title of the book to search for.
 - **Response:** 200 OK with a list of books that match the search criteria.

- **Get Books by Genre**
 - **URL:** GET /api/books/bygenre/{genreId}
 - **Description:** Retrieves books that belong to a specific genre.
 - **Path Parameter:** genreId (Long) - The ID of the genre.
 - **Response:** 200 OK with a list of books in the genre.

- **Remove Genres from Book**
 - **URL:** DELETE /api/books/{bookId}/removeGenres
 - **Description:** Removes genres from an existing book.
 - **Path Parameter:** bookId (Long) - The ID of the book.
 - **Request Body:** List of genre IDs.
 - **Response:** 200 OK with the updated book or 404 Not Found if the book or genre does not exist.

Genres

- **Get All Genres**
 - **URL:** GET /api/genres
 - **Description:** Retrieves a list of all genres.
 - **Response:** 200 OK with a list of genres.
- **Get Genre by ID**
 - **URL:** GET /api/genres/{id}
 - **Description:** Retrieves a genre by its ID.
 - **Path Parameter:** id (Long) - The ID of the genre.
 - **Response:** 200 OK with the genre details or 404 Not Found if the genre does not exist.
- **Create Genre**
 - **URL:** POST /api/genres
 - **Description:** Creates a new genre.
 - **Request Body:** Genre details.
 - **Response:** 201 Created with the created genre.
- **Update Genre**
 - **URL:** PUT /api/genres/{id}
 - **Description:** Updates an existing genre.
 - **Path Parameter:** id (Long) - The ID of the genre.
 - **Request Body:** Updated genre details.
 - **Response:** 200 OK with the updated genre or 404 Not Found if the genre does not exist.
- **Delete Genre**
 - **URL:** DELETE /api/genres/{id}
 - **Description:** Deletes a genre by its ID.
 - **Path Parameter:** id (Long) - The ID of the genre.
 - **Response:** 204 No Content or 404 Not Found if the genre does not exist.

Roles

- **Create Role**
 - **URL:** `POST /api/roles`
 - **Description:** Creates a new role.
 - **Request Body:** Role creation details.
 - **Response:** `201 Created` with the created role.
- **Get Role by ID**
 - **URL:** `GET /api/roles/{id}`
 - **Description:** Retrieves a role by its ID.
 - **Path Parameter:** `id` (Long) - The ID of the role.
 - **Response:** `200 OK` with the role details or `404 Not Found` if the role does not exist.
- **Get All Roles**
 - **URL:** `GET /api/roles`
 - **Description:** Retrieves a list of all roles.
 - **Response:** `200 OK` with a list of roles.
- **Update Role**
 - **URL:** `PUT /api/roles/{id}`
 - **Description:** Updates an existing role.
 - **Path Parameter:** `id` (Long) - The ID of the role.
 - **Request Body:** Updated role details.
 - **Response:** `200 OK` with the updated role or `404 Not Found` if the role does not exist.
- **Delete Role**
 - **URL:** `DELETE /api/roles/{id}`
 - **Description:** Deletes a role by its ID.
 - **Path Parameter:** `id` (Long) - The ID of the role.
 - **Response:** `204 No Content` or `404 Not Found` if the role does not exist.

Store

- **Get All Books**
 - **URL:** `GET /api/store`
 - **Description:** Retrieves a list of all books available in the store.
 - **Response:** `200 OK` with a list of books.
- **Search Books by Title**
 - **URL:** `GET /api/store/search`
 - **Description:** Searches for books by title.
 - **Query Parameter:** `title` (String) - The title of the book to search for.
 - **Response:** `200 OK` with a list of books that match the search criteria.
- **Get Books by Genre**
 - **URL:** `GET /api/store/bygenre/{genreId}`
 - **Description:** Retrieves books that belong to a specific genre.
 - **Path Parameter:** `genreId` (Long) - The ID of the genre.
 - **Response:** `200 OK` with a list of books in the genre.
- **Get Books by Price**
 - **URL:** `GET /api/store/books/by-price`
 - **Description:** Retrieves books based on price criteria.
 - **Query Parameters:**
 - `price` (double) - The price to compare against.
 - `moreThan` (boolean, optional, default `true`) - If `true`, retrieves books with a price greater than the specified value. If `false`, retrieves books with a price less than the specified value.
 - **Response:** `200 OK` with a list of books that meet the criteria.
- **Buy Books**
 - **URL:** `POST /api/store/buybooks`
 - **Description:** Processes a purchase of books.
 - **Request Body:** List of book IDs to be purchased.
 - **Response:** `200 OK` with the purchase details including total price and price with VAT.

Users

- **Create User**
 - **URL:** POST /api/users
 - **Description:** Creates a new user.
 - **Request Body:** User details including roles.
 - **Response:** 201 Created with the created user.
- **Get All Users**
 - **URL:** GET /api/users
 - **Description:** Retrieves a list of all users.
 - **Response:** 200 OK with a list of users.
- **Get User by ID**
 - **URL:** GET /api/users/{id}
 - **Description:** Retrieves a user by its ID.
 - **Path Parameter:** id (Long) - The ID of the user.
 - **Response:** 200 OK with the user details or 404 Not Found if the user does not exist.
- **Get User by Username**
 - **URL:** GET /api/users/username/{username}
 - **Description:** Retrieves a user by username.
 - **Path Parameter:** username (String) - The username of the user.
 - **Response:** 200 OK with the user details or 404 Not Found if the user does not exist.
- **Update User**
 - **URL:** PUT /api/users/{id}
 - **Description:** Updates an existing user.
 - **Path Parameter:** id (Long) - The ID of the user.
 - **Request Body:** Updated user details including roles.
 - **Response:** 200 OK with the updated user.

- **Delete User**

- **URL:** DELETE /api/users/{id}
- **Description:** Deletes a user by its ID.
- **Path Parameter:** id (Long) - The ID of the user.
- **Response:** 204 No Content or 404 Not Found if the user does not exist.

- **Add Role to User**

- **URL:** POST /api/users/{userId}/roles/{roleName}
- **Description:** Assigns a role to a user.
- **Path Parameters:**
 - userId (Long) - The ID of the user.
 - roleName (String) - The name of the role to add.
- **Response:** 200 OK on successful role assignment.

- **Remove Role from User**

- **URL:** DELETE /api/users/{userId}/roles/{roleName}
- **Description:** Removes a role from a user.
- **Path Parameters:**
 - userId (Long) - The ID of the user.
 - roleName (String) - The name of the role to remove.
- **Response:** 204 No Content on successful role removal.

Deployment

As said in the technology stack, this project requires Java 17, so ensure the installation of **JDK 17** and **MAVEN**.

Next, one must run the maven commands:

mvn clean

mvn install

This can be done from the IDE or if needed from the directory in which the pom.xml file is located

Thanks to docker the deployment becomes a lot simpler. After cloning the repository, one can move to the directory in which the compose.yaml files are located and run the command:

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

as such:

```
C:\Users\HP\Documents\Proyectos\proyectosAcademiaJava\Semana5\BookStore>docker-compose -f compose.yaml -f compose-r.yaml up
```

Finally, one can run the BookStoreApplication.java as a java application to run the project locally and access the routes from the port 8080:

http://localhost:8080/

Additionally MySql and RabbitMQ run in their respective ports:

- MySQL: 3306
- RabbitMQ: 5672