Instructions Assignment 2

Elias Bengtsson, Carlo Rashid Galván Urzúa, Maja Hanspers, John Nilsson, Tilde Pennhage

Introduction

This project is a RESTful API built with Spring Boot that allows users to manage a collection of books. The API supports basic CRUD operations, meaning users can create new books, retrieve existing ones, search and filter them, and delete books from the database. Each book stored in the system contains information such as title, description, published year, author, and category. The API follows REST principles and uses standard HTTP methods like **GET**, **POST**, and **DELETE** to perform operations. Proper validation and error handling are implemented to ensure that invalid requests are handled correctly. Since it uses an in-memory database, no external database installation is required, which makes it simple to run and experiment with locally.

Technologies Used

The project is built using several technologies that work together to create a functional REST API. **Spring Boot** is the main framework used to build the application. It simplifies the setup and configuration of Java applications by providing built-in defaults and automatic configuration (Broadcom inc., n.d.). **Spring Web** is used to create REST controllers and handle HTTP requests and responses. It allows us to define endpoints such as GET and POST methods in a clean and structured way (Shrivastava, 2025). **Spring Data JPA** is used to interact with the database. It simplifies database operations by defining repository interfaces instead of writing SQL queries manually (Zewil, 2024). **H2 Database** is used as an in-memory database. It runs inside the application and does not require any separate installation. This makes development and testing easier (GeeksforGeek, 2025). **Maven** is used for dependency management and project building. It handles downloading required libraries and ensures the project compiles correctly (Mishra, 2024).

Building the Project

To build the project, you first need to make sure that *Java 21* is installed on your computer. You can check this by opening a terminal or command prompt and typing java -version. If Java is installed correctly, the version number will be displayed. If not, you need to download and install Java before continuing. If you are using an IDE such as IntelliJ IDEA or Eclipse, you can build the project by opening the project folder in the IDE. The IDE will automatically detect the Maven configuration and download the necessary dependencies. After that, the project is ready to run.

Running the Application

After the project has been built successfully, you can run the application using your IDE, such as IntelliJ IDEA or Eclipse. First, open the project in your IDE. Make sure the project has finished loading and that Maven dependencies have been downloaded. To start the Spring Boot API, locate the main class in the API project. This class contains the main method with the @SpringBootApplication annotation. Click on the "start-button". The application will start, and you should see console output indicating that Spring Boot is running. This means the API is now running and ready to accept requests at http://localhost:8080. Open the console client application project in your IDE as well. Locate the “Main class” and run it in the same way. Make sure that the Spring Boot API is already running before starting the console client, otherwise the client will not be able to connect. When the console client starts, you will see a menu displayed in the console. You can then select options to retrieve books, create a new book, delete a book, or perform other operations. Once both applications are running correctly, you can test all CRUD operations through the console menu or by using a tool such as Postman to send HTTP requests directly to the API.

Project structure

The project is divided into two parts, which are the Spring Boot API and the console client. The API contains The main application class (BooksApplication.java), The Book class, The Category enum, The BookRepository interface, The BookController class, and The application.properties file. The API handles logic and database storage.
The console client project contains The Main class with the switch menu, and the HTTP requests to communicate with the API. It also contains the pom.xml file, determining the necessary dependencies in the project. The client does not store data. It only sends requests and displays responses.

Database Configuration

This project uses an in-memory database called H2. This means that the database runs only while the application is running. When you stop the application, all stored data is deleted automatically. The database is configured in the application.properties file. Because it is in-memory, you do not need to install any database software manually. Spring Boot automatically creates the database when the application starts. If you want to view the database in a browser, you can open: http://localhost:8080/h2. When the H2 console page appears, use these settings:

- JDBC URL: jdbc:h2:mem:bookdb
- Username: sa
- Password: (leave empty)

Click Connect. You will then see the database tables and can run SQL queries if needed.

API Endpoints and CRUD

This API follows the principles of CRUD, which stands for Create, Read, Update, and Delete. In this project, the API provides endpoints that allow users to create, retrieve, and delete books stored in the database. An update operation is not required in this assignment, but the others are fully implemented. Using Postman, the **Create** operation is handled by the POST endpoint. This allows the user to send book information in JSON format, and the application saves it to the database. The ID is generated automatically, ensuring each book has a unique primary key. The **Read** operations are handled by multiple GET endpoints, for example to retrieve all books in the database, a book with a specific ID, a book with a specific word in the title, or books belonging to a specific category. These endpoints allow users to access data in different ways depending on what they need. The **Delete** operation is handled by the DELETE endpoint. This removes a book from the database based on its ID. If the book does not exist, the API returns an appropriate error response. Together, these endpoints provide a complete RESTful structure for managing the book database. Each operation uses the correct HTTP method and follows REST conventions, making the API consistent, predictable, and easy to test using tools like Postman.

Testing the API

After the application is running, you need to test that all endpoints work correctly. The two main ways to do this is either using your console client or using a tool like Postman. If you are using the console client, simply run the Main class of the client project while the Spring Boot API is running. A menu will appear in the console. From there, you can choose options to retrieve all books, search by ID, search by title, filter by category, create a new book, or delete a book. Each option sends a request to the API and displays the result. If you are using Postman, open Postman and create a new request. Make sure the API is running first. Then use the following examples:

- To retrieve all books, select the GET method and enter: http://localhost:8080/books
- To retrieve a book by ID, use: http://localhost:8080/books/1, where 1 represents the specific book ID that you want to retrieve.
- To retrieve books by title, use: http://localhost:8080/books/title?title=java. Replace "java" with any word you want to search for.
- To retrieve books by category, use: http://localhost:8080/books/category?category=IT. Replace “IT” with the category you want to search for.
- To create a new book, choose the POST method and use http://localhost:8080/books. Then go to the Body tab, choose raw, select JSON format, and enter something like:

```
{
 "title": "Harry Potter and the Goblet of Fire",
"description": "Harry has problem getting the golden egg to open",
"publishedYear": "2011",
"author": "J.K. Rowling",
"category": "OTHER"
}
```

Click Send. If everything is correct, you should receive status 201 Created and the saved book with a generated ID.

- To delete a book, choose DELETE and use: http://localhost:8080/books/1, where 1 represents the specific book ID that you want to delete. If successful, you should receive status 204 No Content.

If you run into any errors, check that:

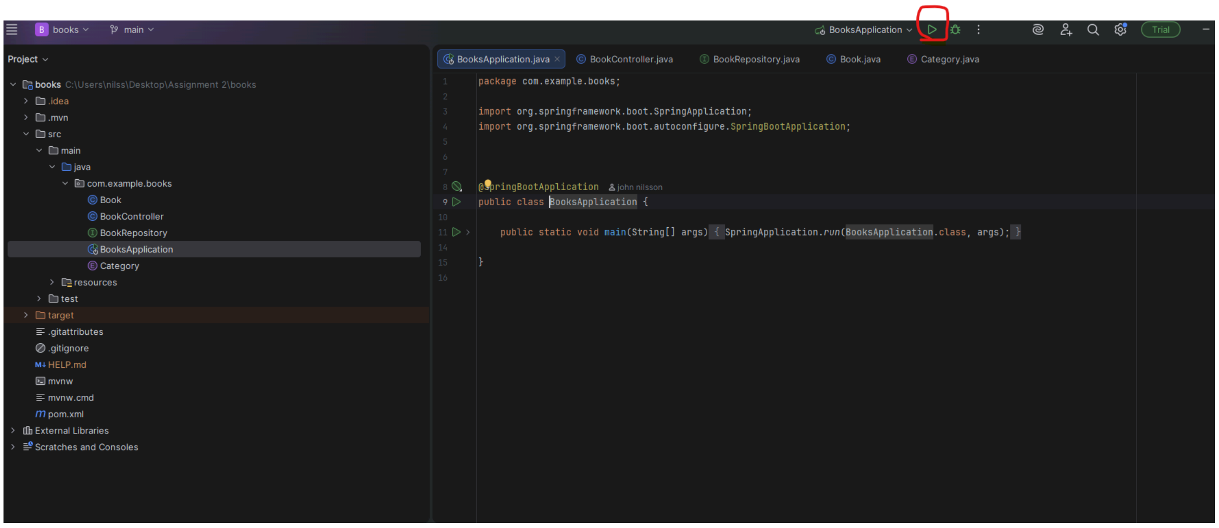
- The API is running
- The URL is correct
- Fields like “title” and “author” are not empty

Stopping the application

To stop the application in IntelliJ or Eclipse, click the red Stop button in the console window. This will shut down the Spring Boot server. Since the database is in-memory, all stored data will be deleted when the application stops. The next time you start it, the database will be empty again.

Step by step How to run:

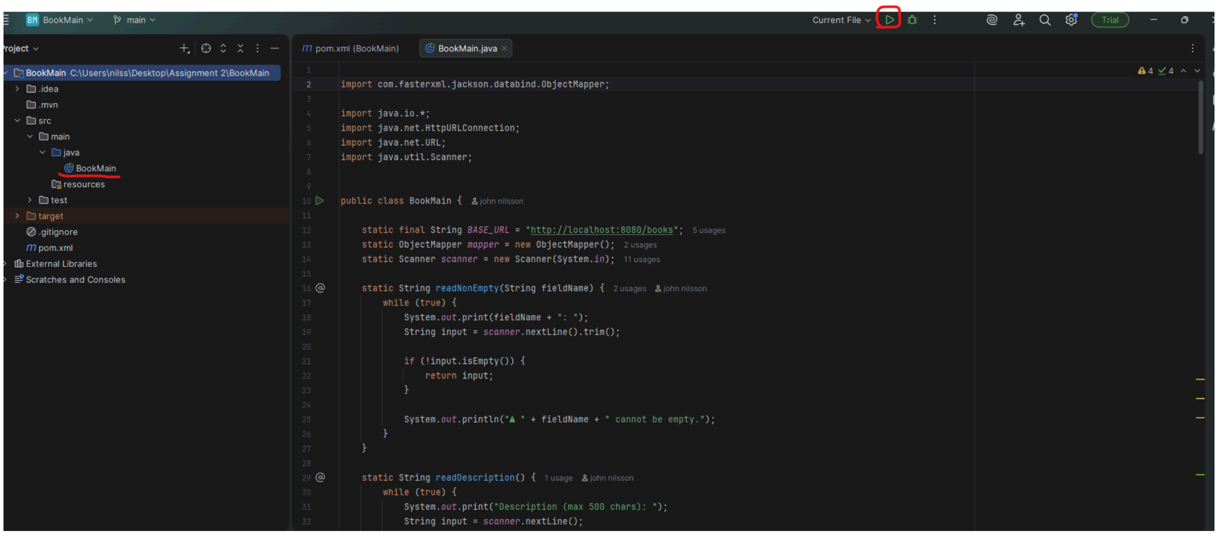
1. In intelliJ open the project with folder “books” and run BooksApplication class.



```java
package com.example.books;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;



@SpringBootApplication  john nilsson
public class BooksApplication {

    public static void main(String[] args) { SpringApplication.run(BooksApplication.class, args); }

}
```

This will setup the api.

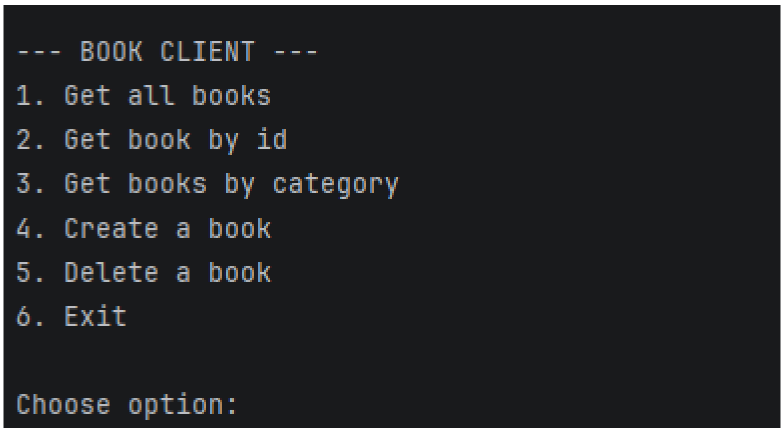
2. In a new instance of IntelliJ open the folder BookMain as a project and run the BookMain class.



```java

import com.fasterxml.jackson.databind.ObjectMapper;

import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Scanner;


public class BookMain {

    static final String BASE_URL = "http://localhost:8080/books";
    static ObjectMapper mapper = new ObjectMapper();
    static Scanner scanner = new Scanner(System.in);

    static String readNonEmpty(String fieldName) {
        while (true) {
            System.out.print(fieldName + ": ");
            String input = scanner.nextLine().trim();

            if (!input.isEmpty()) {
                return input;
            }

            System.out.println("⚠ " + fieldName + " cannot be empty.");
        }
    }

    static String readDescription() {
        while (true) {
            System.out.print("Description (max 500 chars): ");
            String input = scanner.nextLine();
```

This will start the console application which we use to handle CRUD operations with the API

3. Use the options in the console menu to add/delete or search books



```
--- BOOK CLIENT ---
1. Get all books
2. Get book by id
3. Get books by category
4. Create a book
5. Delete a book
6. Exit

Choose option:
```

References

Broadcom Inc. (n.d.). *Spring Boot*. https://docs.spring.io/spring-boot/index.html

GeeksforGeeks. (2025, 23 July). *Spring Boot with H2 Database*. https://www.geeksforgeeks.org/springboot/spring-boot-with-h2-database/

Mishra, H. (2024, 6 December). *Maven Guide for Beginners*. DEV Community. https://dev.to/harshm03/maven-guide-for-beginners-2083

Shrivastava, A. (2025, 25 February). *Introduction to Spring REST*. DEV Community. https://dev.to/ayshriv/introduction-to-spring-rest-3nhl

Zewil, A. (2024, 24 november). *Unlocking the Power of Spring Data JPA*. Sumerge. https://www.sumerge.com/spring-data-jpa/