# Exercise 5: Configuring the Spring IoC Container

## Introduction

This exercise focuses on configuring the Spring Inversion of Control (IoC) container using an XML-based configuration. The goal is to develop a simple Library Management application where dependencies between beans (e.g., BookService and BookRepository) are managed by the Spring container.

## Objective

* Create and configure an XML-based Spring application context.
* Define and wire beans using the XML configuration.
* Demonstrate dependency injection via setter methods.
* Load the application context and test the configuration with a simple main class.

## Tools Required

* Java 17 or above
* VS Code with Spring extensions or any Java IDE
* Spring Core dependency (included via Maven or manually)

## Step-by-Step Implementation

### Step 1: applicationContext.xml

### 

### Step 3: Book.java (Model)

### 

### Step 4: BookRepository.java

### Step 5: BookService.java

### 

### Step 6: LibraryApp.java (Main Class)

## Expected Output

## Conclusion

This exercise demonstrates the use of Spring IoC container with XML configuration. By defining and wiring beans externally, Spring promotes loose coupling and easier testability. You also practiced setting up a Spring project with POJOs, repositories, services, and context loading.

**Exercise 7: Implementing Constructor and Setter Injection**

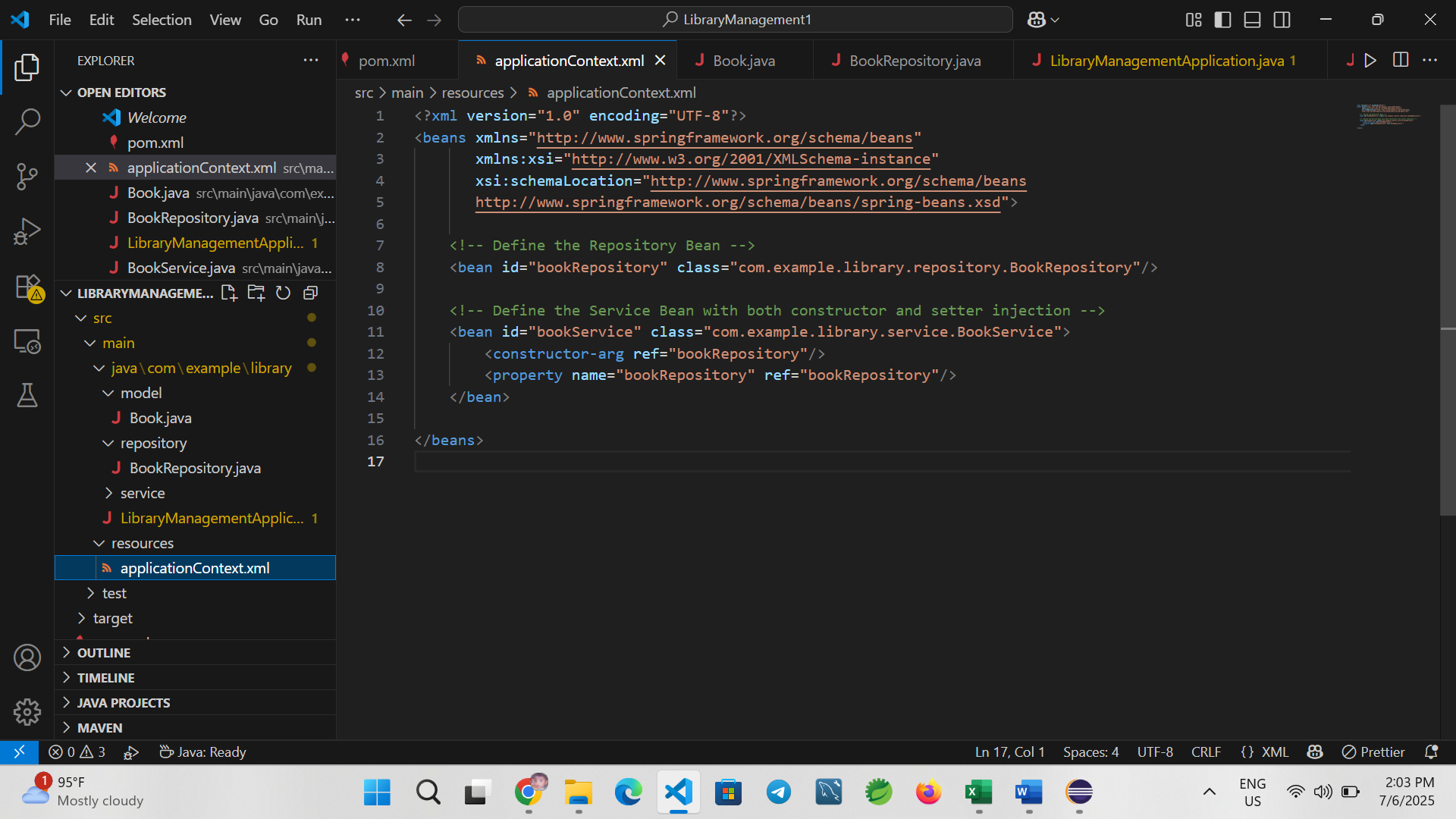
**Introduction:** In this exercise, we explore two essential methods of dependency injection in the Spring Framework: constructor injection and setter injection. This practice helps ensure flexible and loosely coupled code, allowing better testability and control over bean initialization. We’ll implement this in the context of a simple Library Management application.

**Objective:** Demonstrate how to configure and use constructor and setter injection using Spring’s XML configuration in a Java-based application.

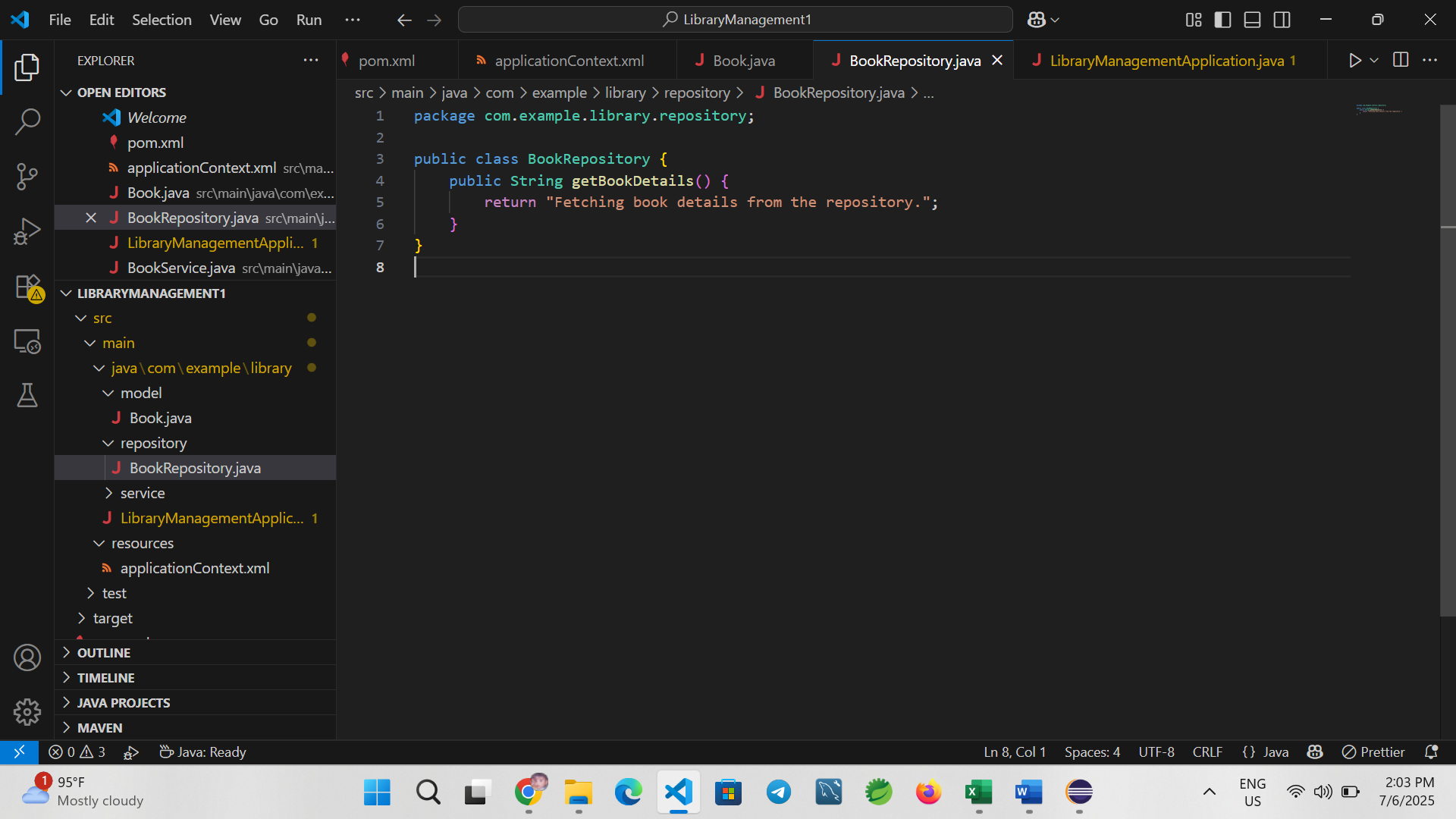
**Step-by-Step Implementation:**

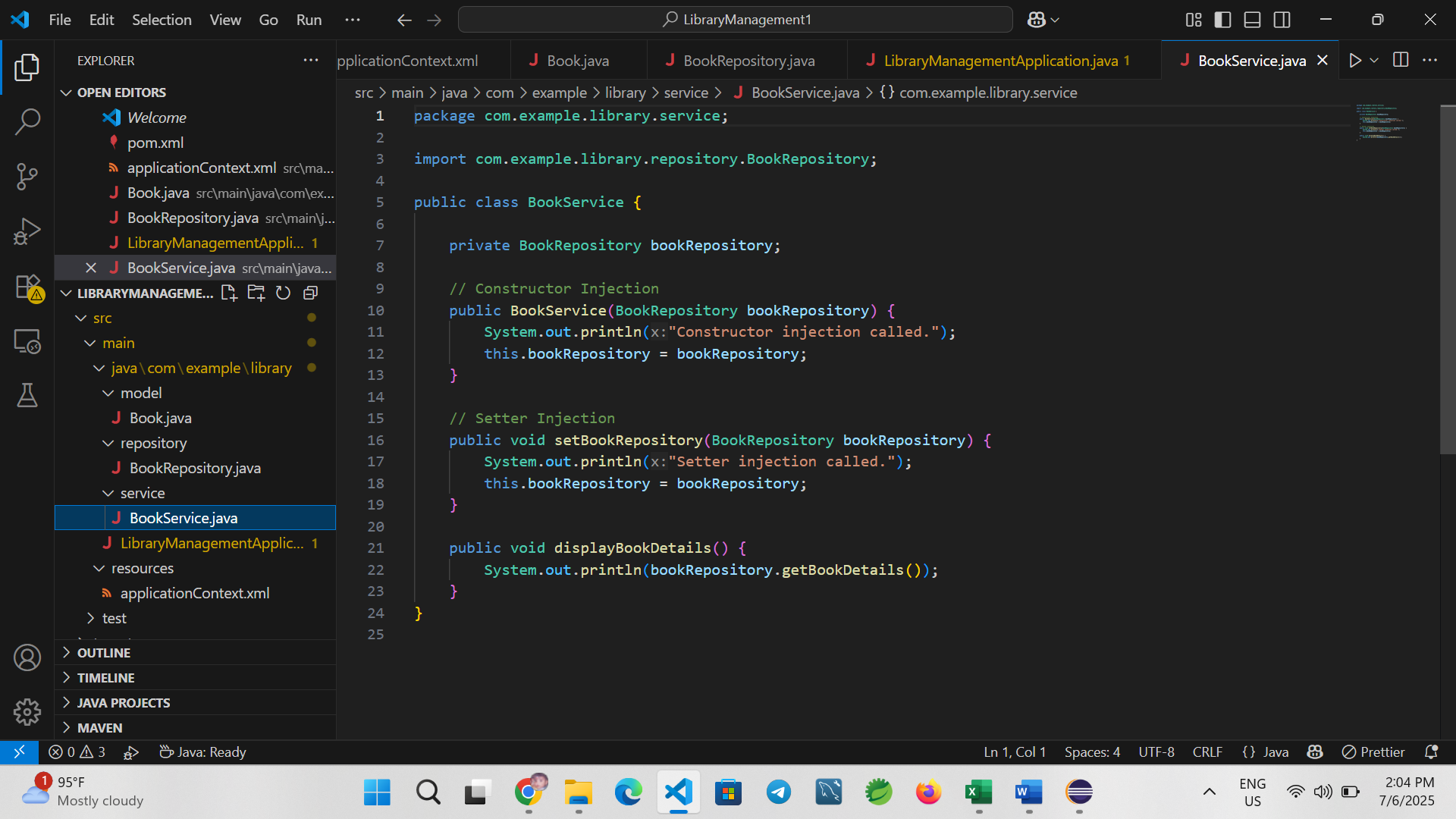
**1. Create a Spring Configuration File (applicationContext.xml):**

* Located under src/main/resources
* It defines the bookRepository bean and the bookService bean with both constructor and setter injection

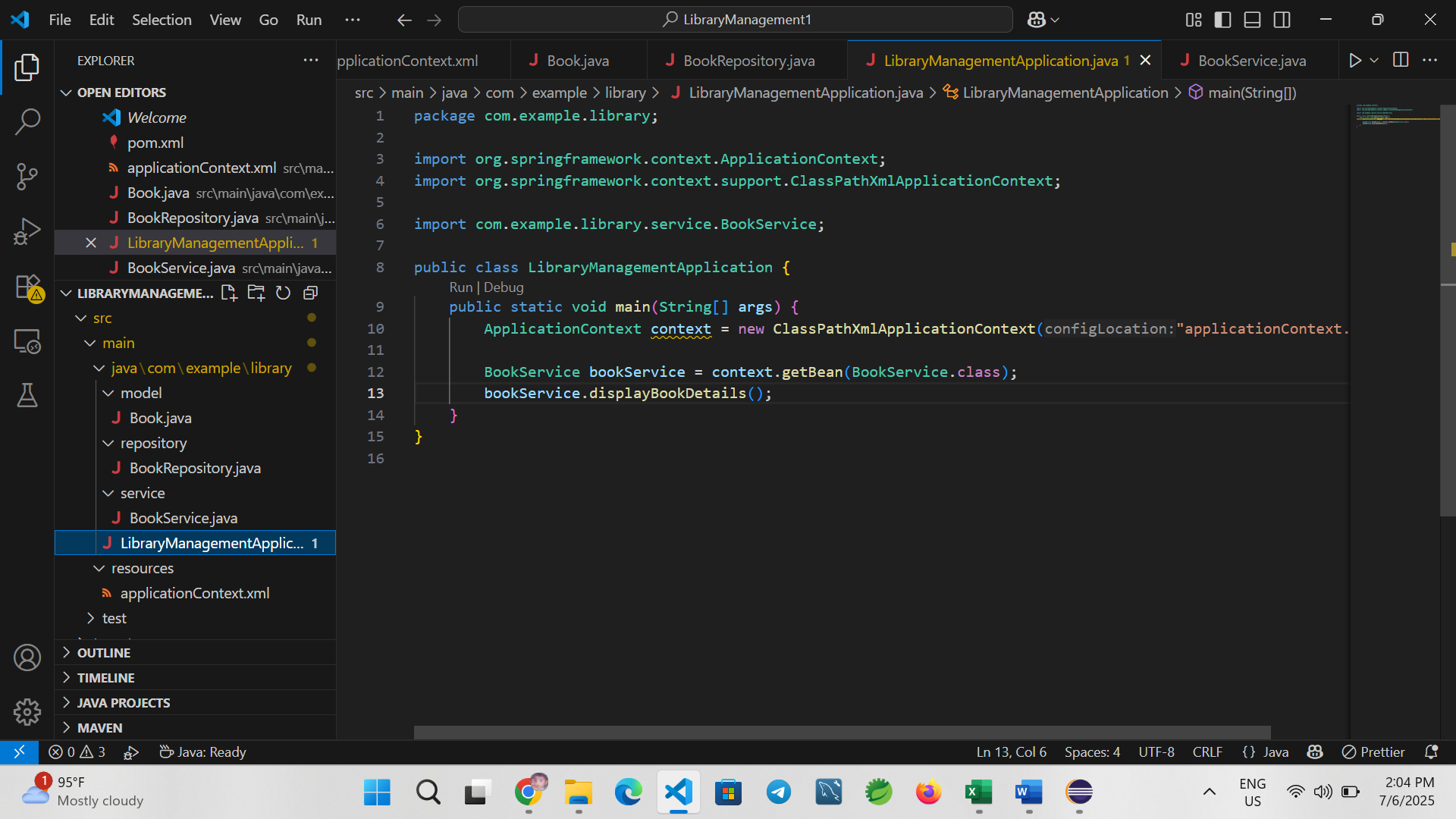


**2. Create the BookRepository Class:**

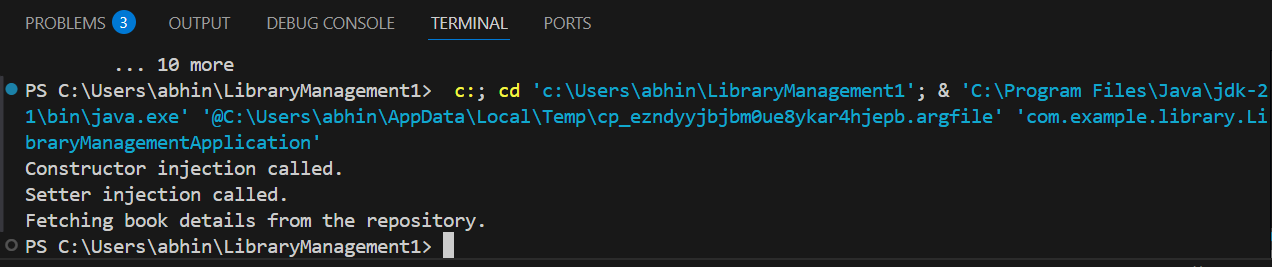
**3. Create the BookService Class:**



**4. Create the Main Class (LibraryManagementApplication.java):**



**Output:**



**Conclusion:** This exercise demonstrated how to configure and use both constructor and setter injection using Spring XML configuration. Constructor injection ensures that required dependencies are not missed during object creation, whereas setter injection provides more flexibility in optional dependencies and configuration. Using both injection methods enhances the robustness and configurability of Spring-based applications.

Exercise 9: Creating a Spring Boot Application for Library Management System

Introduction

This exercise involves developing a Spring Boot application to manage a library system. The application supports basic CRUD operations (Create, Read, Update, Delete) for book records. By using Spring Boot, the setup and configuration are simplified, allowing for faster development and deployment.

Objective

Create a fully functional RESTful web service for managing books using Spring Boot, Spring Data JPA, and H2 in-memory database.

Step 1: Create Spring Boot Project

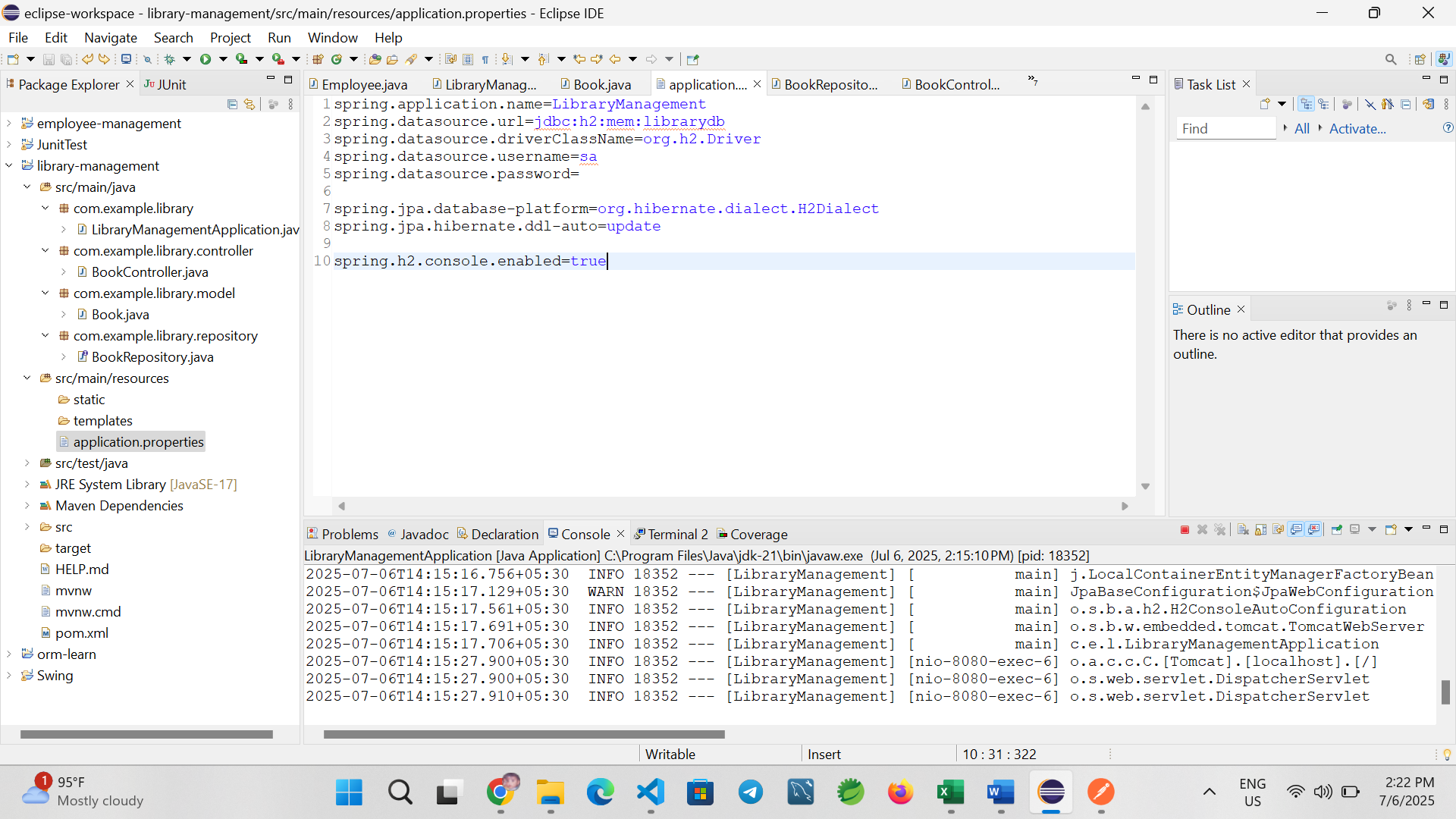
* Use [Spring Initializr](https://start.spring.io/)
* Project Name: LibraryManagement
* Project Type: Maven
* Language: Java
* Spring Boot Version: 3.0+ (latest stable)

Dependencies:

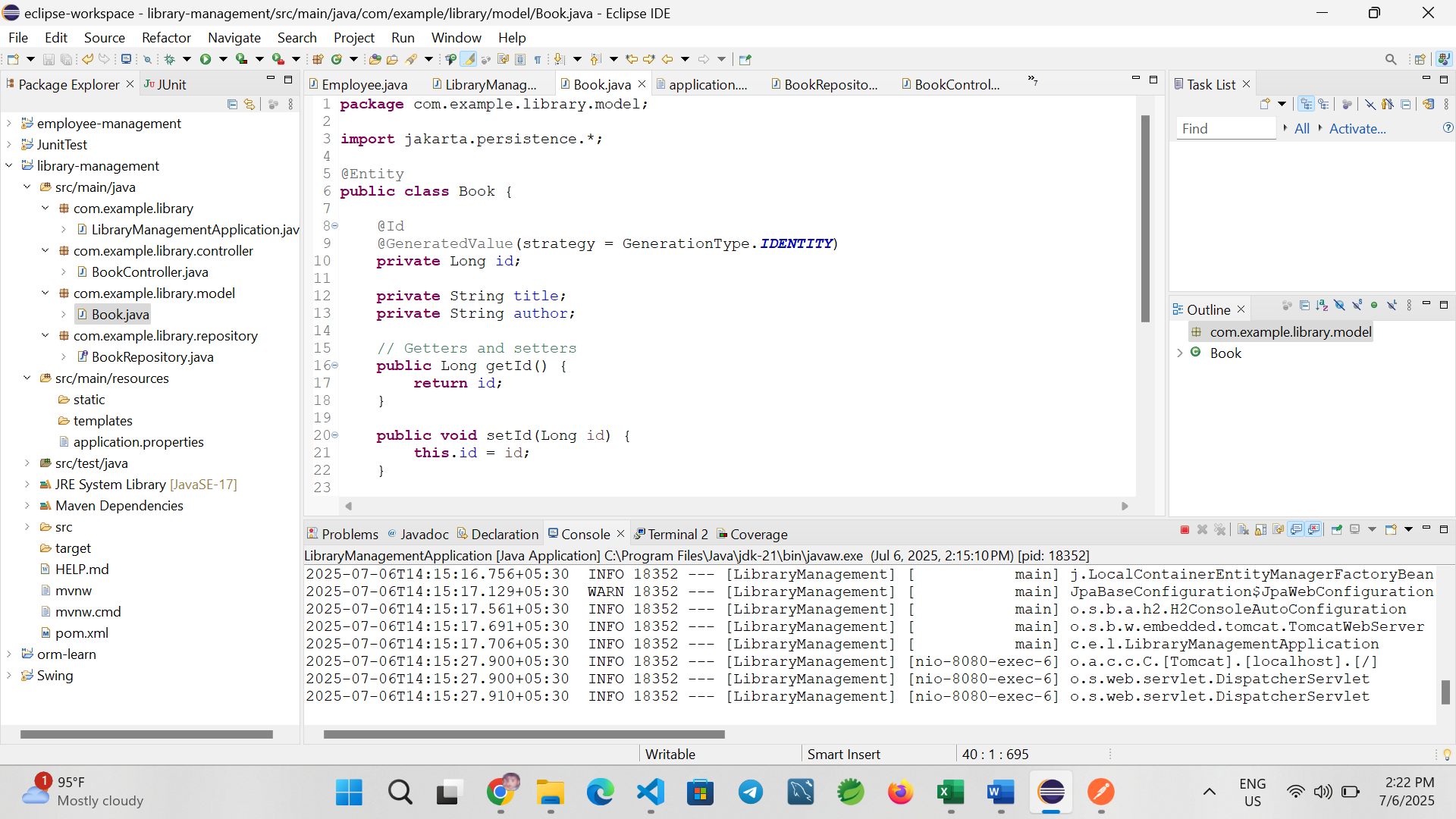
* Spring Web
* Spring Data JPA
* H2 Database

Download the project and import it into your IDE (e.g., VS Code or Eclipse).

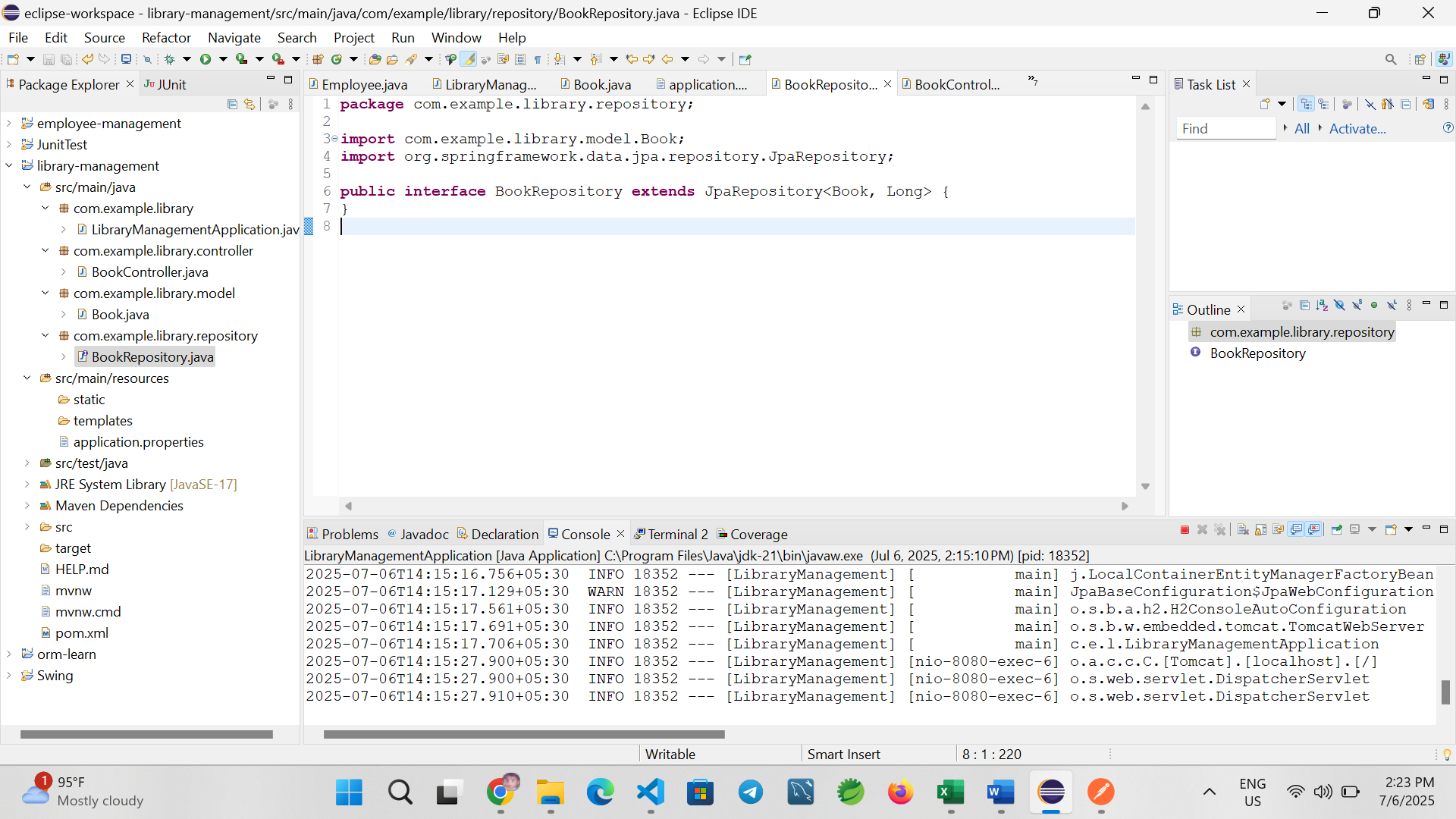
Step 2: Define application.properties



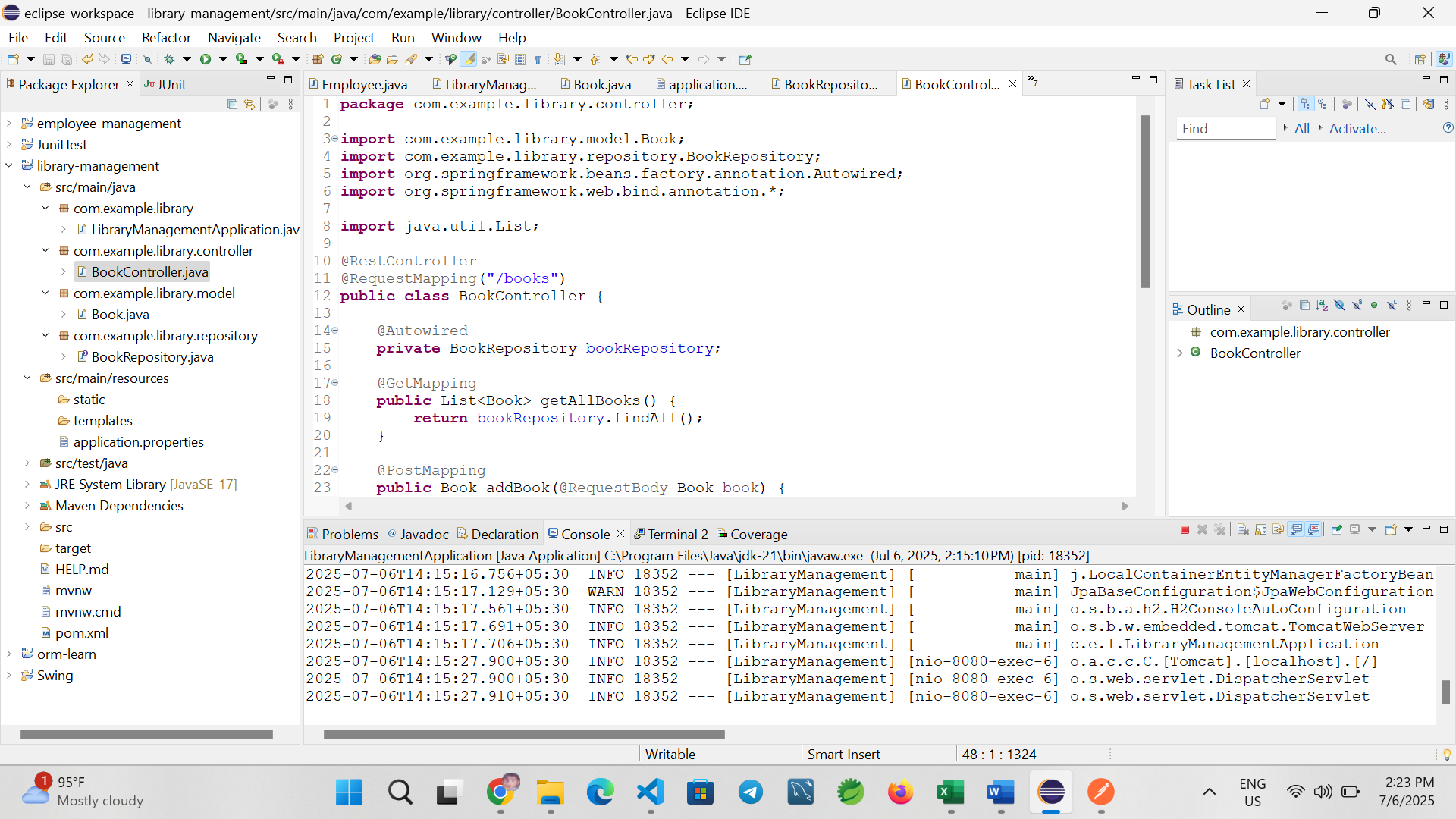
Step 4: Create Book Entity



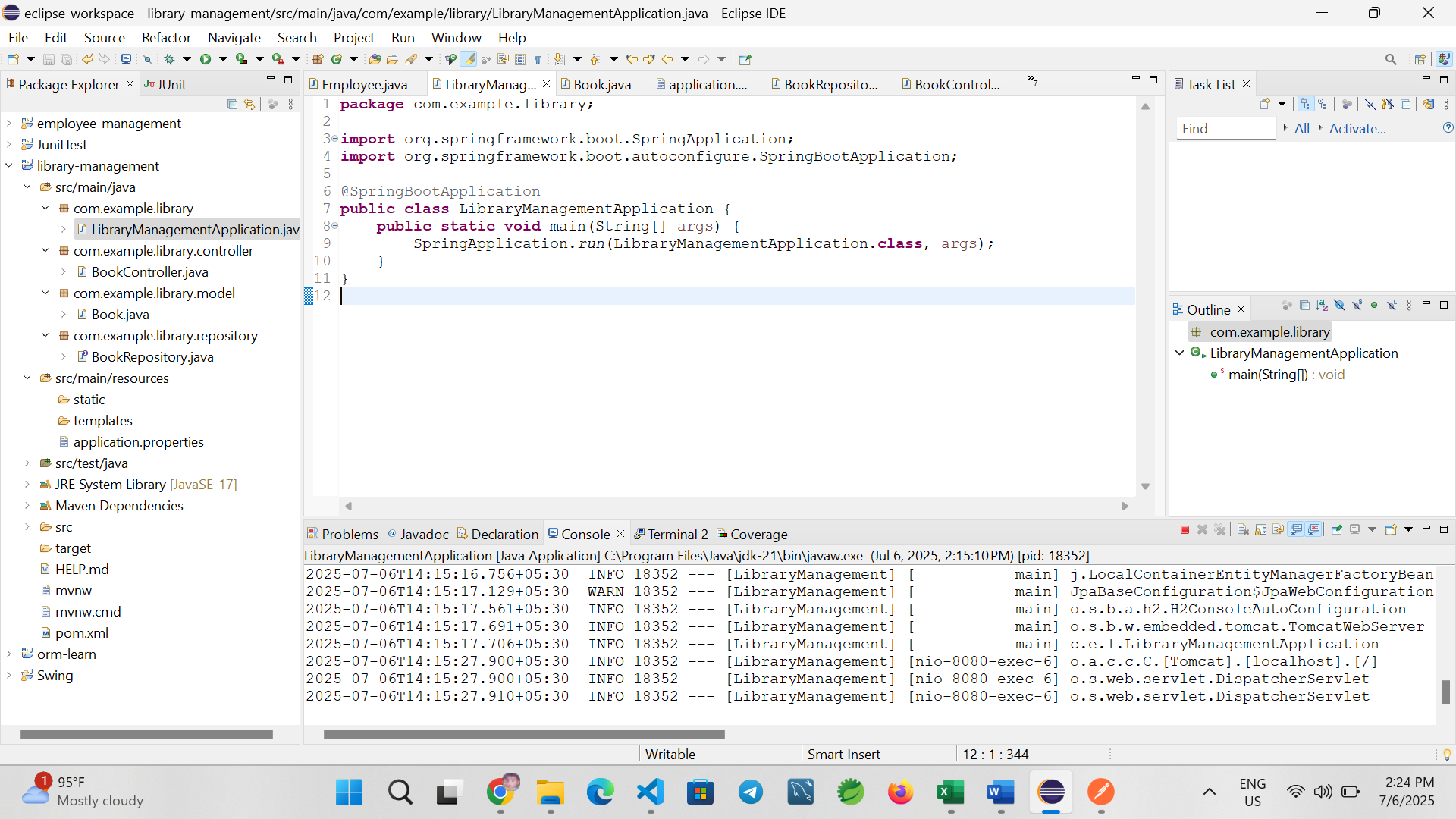
Step 5: Create Book Repository



Step 6: Create Book Controller



Step 7: Main Application



Step 8: Test Using Postman

Use the following endpoints to test your application:

| Method | Endpoint | Description |
| --- | --- | --- |
| POST | /books | Add a new book |
| GET | /books | Get all books |
| GET | /books/{id} | Get book by ID |
| PUT | /books/{id} | Update book by ID |
| DELETE | /books/{id} | Delete book by ID |

Example JSON for POST/PUT:



Conclusion

In this exercise, we successfully built a Spring Boot application to manage books in a library. The app uses Spring Web for RESTful endpoints, Spring Data JPA for database operations, and H2 for in-memory testing. Postman was used to test each endpoint to ensure full CRUD functionality.