# **Exercise 9: Creating a Spring Boot Application**

## **Scenario**

Create a Spring Boot application for a Library Management System with CRUD operations using:

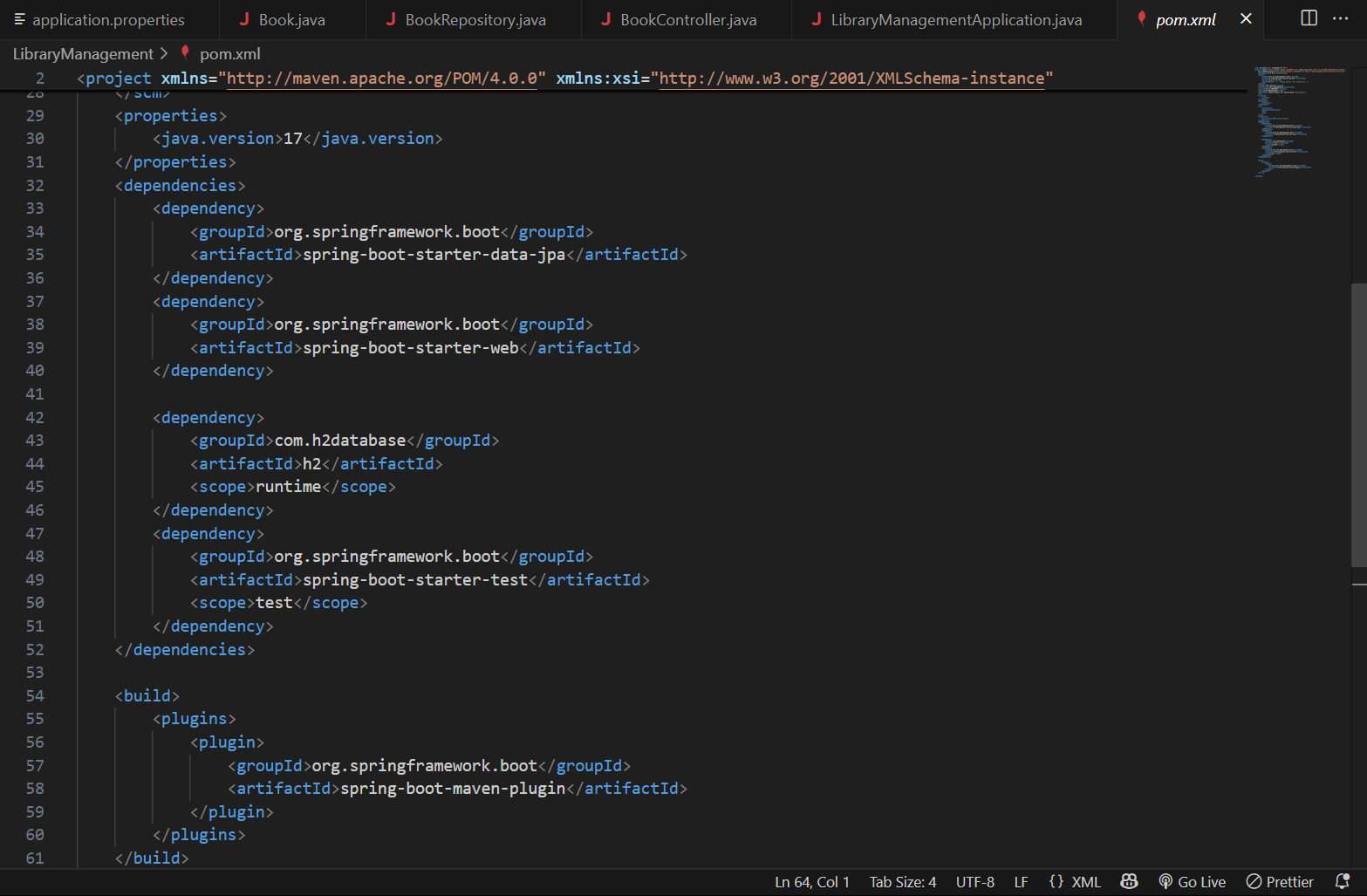
* Spring Web
* Spring Data JPA
* H2 Database

## **Step 1: Project Structure**

* Generated using Spring Initializr
* Project Name: LibraryManagement
* Base Package: com.library
* Dependencies: Spring Web, Spring Data JPA, H2 Database

## **Step 2: Final pom.xml**

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.5.3</version>  
 <relativePath/>  
</parent>  
  
<groupId>com.library</groupId>  
<artifactId>LibraryManagement</artifactId>  
<version>0.0.1-SNAPSHOT</version>  
<name>LibraryManagement</name>  
  
<properties>  
 <java.version>17</java.version>  
</properties>  
  
<dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>  
  
<build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
</build>

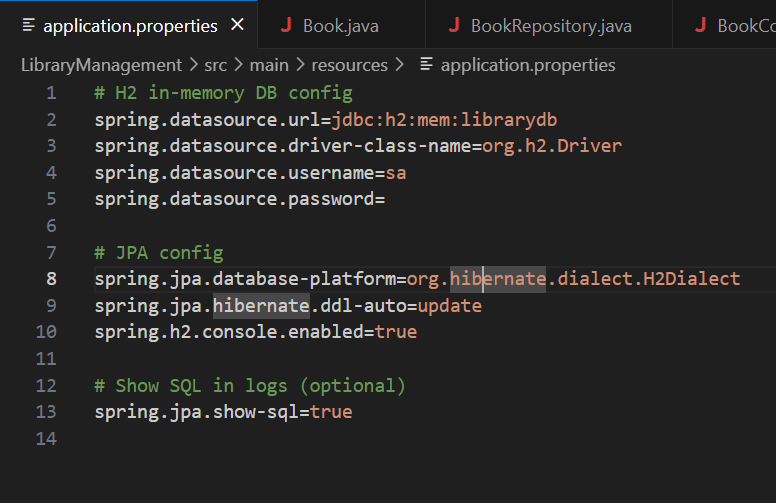


## **Step 3: application.properties**

**File: src/main/resources/application.properties**

spring.datasource.url=jdbc:h2:mem:librarydb  
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
spring.jpa.hibernate.ddl-auto=update  
spring.h2.console.enabled=true  
spring.jpa.show-sql=true

Configures an in-memory H2 DB.

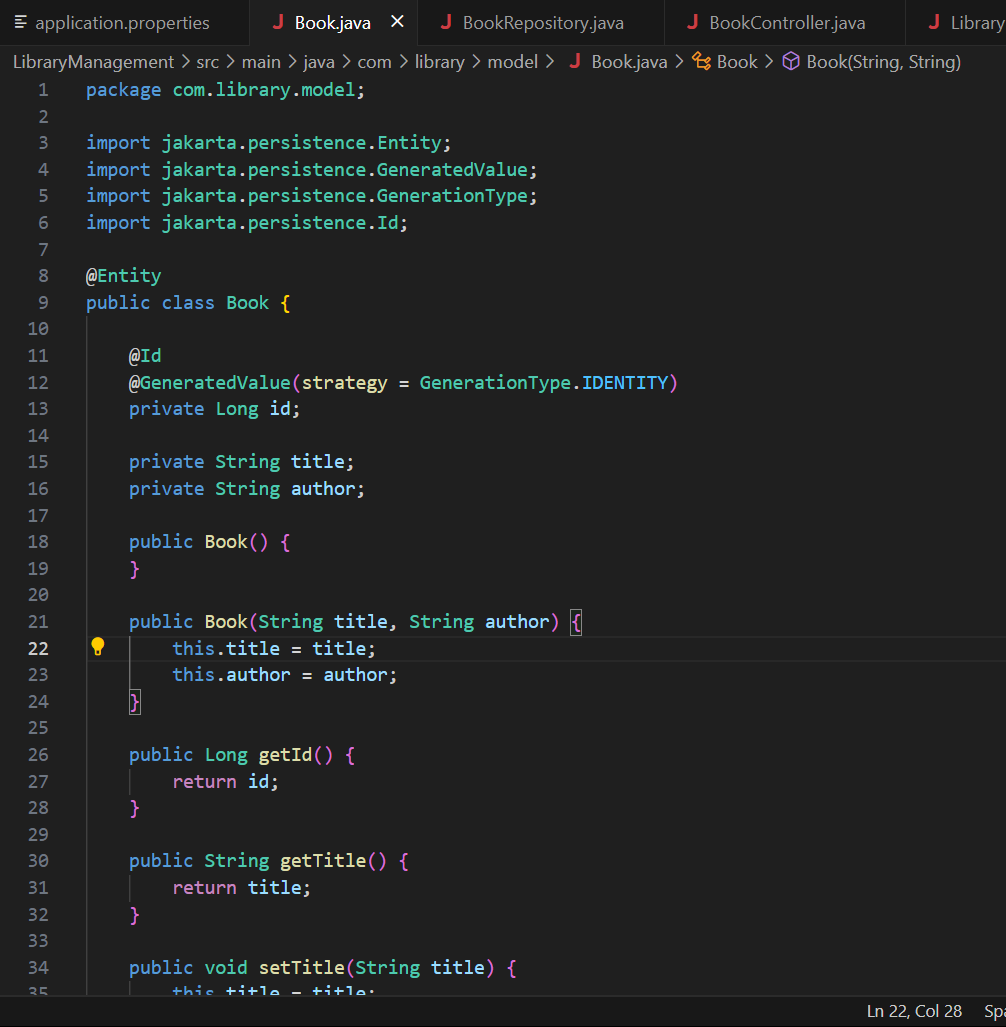


## **Step 4: Entity**

**File: src/main/java/com/library/model/Book.java**

package com.library.model;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
@Entity  
public class Book {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 private String title;  
 private String author;  
  
 public Book() {}  
  
 public Book(String title, String author) {  
 this.title = title;  
 this.author = author;  
 }  
  
 public Long getId() { return id; }  
  
 public String getTitle() { return title; }  
 public void setTitle(String title) { this.title = title; }  
  
 public String getAuthor() { return author; }  
 public void setAuthor(String author) { this.author = author; }  
}

Maps to a BOOK table automatically.

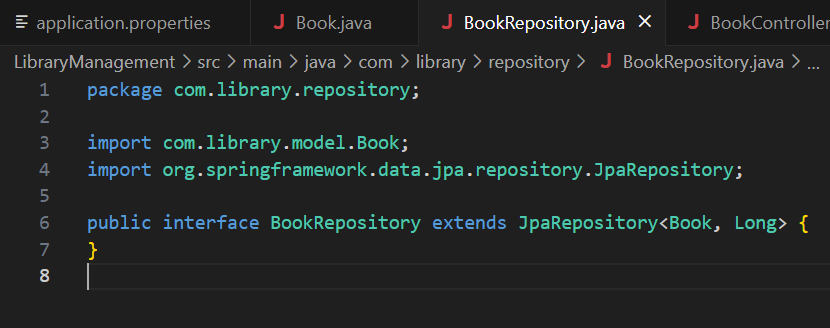


## **Step 5: Repository**

**File: src/main/java/com/library/repository/BookRepository.java**

package com.library.repository;  
  
import com.library.model.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BookRepository extends JpaRepository<Book, Long> {  
}

Provides CRUD operations.

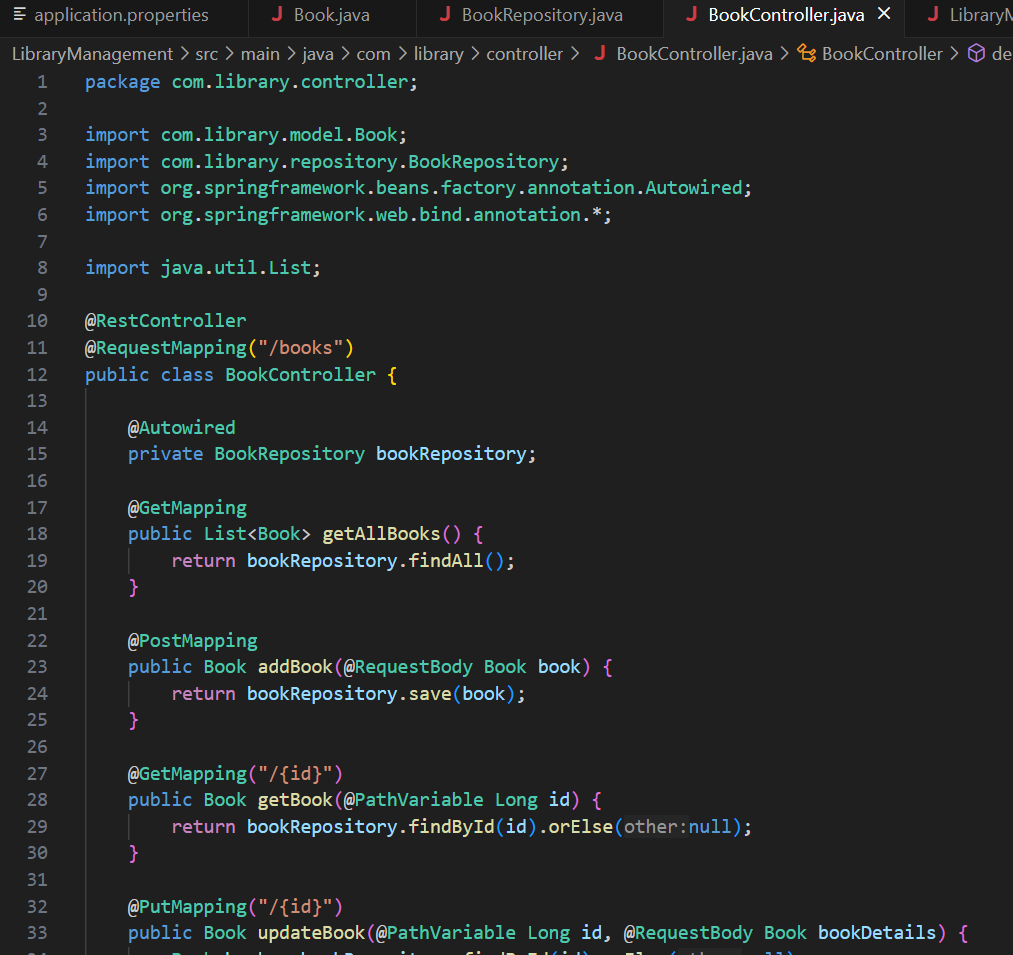


## **Step 6: REST Controller**

**File: src/main/java/com/library/controller/BookController.java**

package com.library.controller;  
  
import com.library.model.Book;  
import com.library.repository.BookRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/books")  
public class BookController {  
  
 @Autowired  
 private BookRepository bookRepository;  
  
 @GetMapping  
 public List<Book> getAllBooks() {  
 return bookRepository.findAll();  
 }  
  
 @PostMapping  
 public Book addBook(@RequestBody Book book) {  
 return bookRepository.save(book);  
 }  
  
 @GetMapping("/{id}")  
 public Book getBook(@PathVariable Long id) {  
 return bookRepository.findById(id).orElse(null);  
 }  
  
 @PutMapping("/{id}")  
 public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {  
 Book book = bookRepository.findById(id).orElse(null);  
 if (book != null) {  
 book.setTitle(bookDetails.getTitle());  
 book.setAuthor(bookDetails.getAuthor());  
 return bookRepository.save(book);  
 }  
 return null;  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable Long id) {  
 bookRepository.deleteById(id);  
 }  
}

Handles full CRUD via REST.

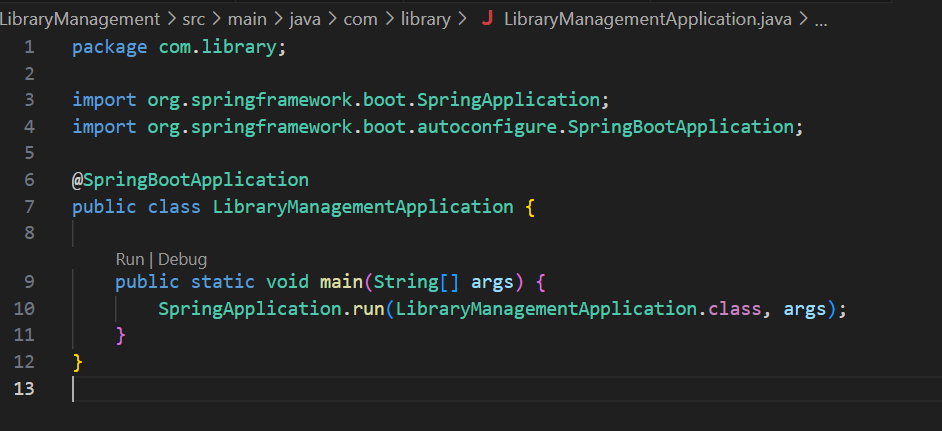


## **Step 7: Main Class**

**File: src/main/java/com/library/LibraryManagementApplication.java**

package com.library;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class LibraryManagementApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.run(LibraryManagementApplication.class, args);  
 }  
}

Runs the full app with embedded Tomcat.



## **Step 8: Postman Examples**

Example 1: Add a Book (POST)

* URL: http://localhost:8080/books
* Method: POST
* Body (JSON):

{  
 "title": "Spring Boot Handbook",  
 "author": "Jane Doe"  
}

Example 2: Get All Books (GET)

* URL: http://localhost:8080/books

Example 3: Get Single Book (GET)

* URL: http://localhost:8080/books/1

Example 4: Update Book (PUT)

* URL: http://localhost:8080/books/1
* Body (JSON):

{  
 "title": "Updated Book",  
 "author": "Updated Author"  
}

Example 5: Delete Book (DELETE)

* URL: http://localhost:8080/books/1

H2 console: <http://localhost:8080/h2-console>  
JDBC URL: jdbc:h2:mem:librarydb

## Logs Proof

Example logs:

Tomcat started on port 8080  
Hibernate: insert into book ...  
Hibernate: select ...  
Hibernate: update book set ...

This confirms your API is fully working!

