Library Book Catalog API

Problem Statement

Build a simple Java-based RESTful API for managing a library's book catalog.

Features

- Add a new book
- Retrieve the list of all books
- Get book details by its ID
- Delete a book by ID
- Update availability status of a book

Technical Requirements

- Java 8+
- Object-oriented design and encapsulation
- In-memory data structure (Map, List)
- REST-style endpoints using Spring Boot
- Input validation (e.g., `title` should not be empty)

Bonus Features Implemented

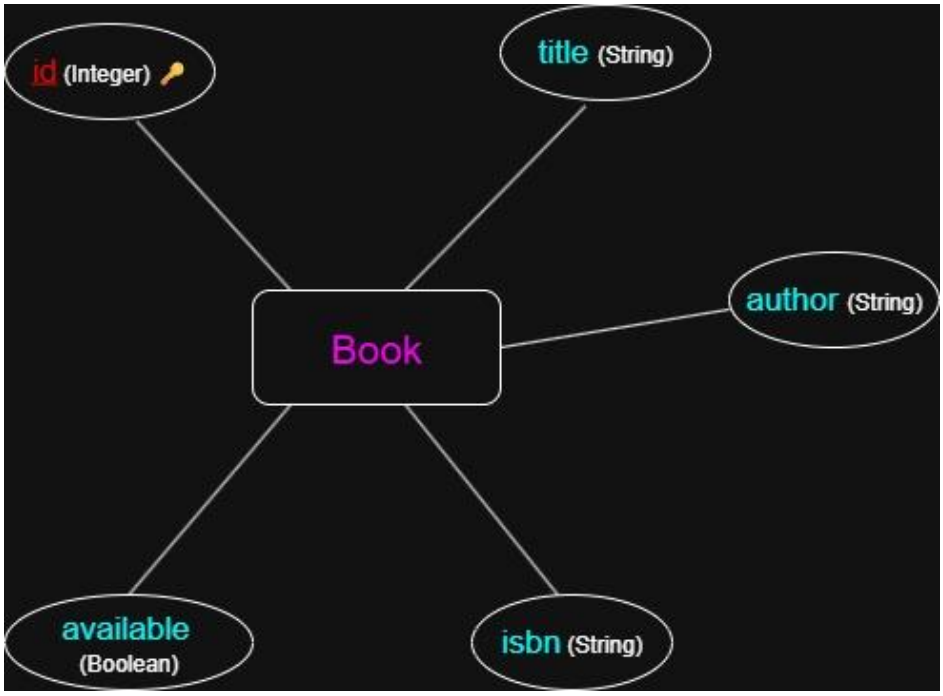
- Spring Boot used to expose REST endpoints
- Postman collection for API testing
- GitHub repository with README and sample requests

Developer Info

Developed by: Darshan
Tools: Spring Boot, Postman, STS, GitHub
Git repo - https://github.com/darshanchaudharii/Library_Management_Books_Catelog_Backend_Server-.git
Postman Shared link - https://restless-rocket-398046.postman.co/workspace/My-Workspace~d611aeb1-7e53-402e-8557-fca0c9eaf520/collection/45915590-141ca865-08a1-49b2-8588-ac7aeda002e4?action=share&creator=45915590

```
com.books_catalog.main/
├── src/
│   └── main/
│       ├── java/
│       │   └── com/books_catelog/main/
│       │       ├── entity/
│       │       │   └── Book.java
│       │       ├── controller/
│       │       │   └── BookController.java
│       │       ├── repository/
│       │       │   └── BookRepository.java
│       │       ├── service/
│       │       │   └── BookService.java
│       │       ├── service/impl/
│       │       │   └── BookServiceImpl.java
│       │       └── LibraryApiApplication.java
│       └── resources/
│           └── application.properties
├── pom.xml
```

Entity Diagram -



1. **Title:** `Book.java (Entity Class)`

```java
package com.books_catelog.main.entity;

import java.util.concurrent.atomic.AtomicInteger;

import jakarta.validation.constraints.NotBlank;
/*
  Represents one Book in our catalog.
 */
public class Book {
    private static final AtomicInteger COUNTER = new AtomicInteger();
 // unique identifier
    private Integer id;
 // If someone tries to create a book without a title validation will fail automatically
    @NotBlank(message = "Title is required")
    private String title;
 // If someone tries to create a book without the author name validation will fail automatically
    @NotBlank(message = "Author is required")
    private String author;

    private String isbn;

    // Default value is true
    private Boolean available = true;


    // Constructor
    public Book(String title, String author, String isbn) {
     // Generates unique ID using AtomicInteger
        this.id = COUNTER.incrementAndGet();
        this.title = title;
        this.author = author;
        this.isbn = isbn;
    }

    // Getters and setters
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = 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;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public Boolean getAvailable() {
        return available;
    }

    public void setAvailable(Boolean available) {
        this.available = available;
    }
}

```

2. **Title:** `BookController.java (Controller Class)`

```java
package com.books_catelog.main.controller;

import com.books_catelog.main.entity.Book;
import com.books_catelog.main.service.BookService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.Map;
/*
 Simple console menu to manage books in the library.
 */

@RestController
@RequestMapping("/books")
public class BookController {

    private final BookService bookService;

    public BookController(BookService bookService) {
        this.bookService = bookService;
    }

    // Add a new book
    @PostMapping
    public ResponseEntity<Book> addBook(@jakarta.validation.Valid @RequestBody Book book) {
        Book savedBook = bookService.addBook(book);
        return ResponseEntity.ok(savedBook);
    }

    // Get all books
    @GetMapping
    public ResponseEntity<Map<Integer, Book>> getAllBooks() {
        Map<Integer, Book> books = bookService.getAllBooks();
        return ResponseEntity.ok(books); // Returns the list of all books
    }

    // Get a book by ID
    @GetMapping("/{id}")
    public ResponseEntity<Book> getBookById(@PathVariable Integer id) {
        Book book = bookService.getBookById(id);
        return (book != null) ? ResponseEntity.ok(book) : ResponseEntity.notFound().build();
    }

    // Delete a book by ID
    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteBook(@PathVariable Integer id) {
        bookService.deleteBook(id);
        return ResponseEntity.noContent().build();
    }

    // Update book availability
    @PatchMapping("/{id}/availability")
    public ResponseEntity<Void> updateAvailability(@PathVariable Integer id, @RequestParam Boolean available) {
        bookService.updateAvailability(id, available);
        return ResponseEntity.noContent().build();
    }
}
```

3. **Title:** `BookRepository.java (Repository Class)`

```java
package com.books_catelog.main.repository;

import com.books_catelog.main.entity.Book;
import org.springframework.stereotype.Repository;

import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

/**
 * Inmemory store for Book objects.
 */
@Repository
public class BookRepository {
    // holds books
    private final Map<Integer, Book> books = new ConcurrentHashMap<>();

    // Save a book in the repository
    public Book save(Book book) {
        books.put(book.getId(), book);
        return book;
    }

    // Get all books from the repository
    public Map<Integer, Book> findAll() {
        return books;
    }

    // Find a book by its ID
    public Book findById(Integer id) {
        return books.get(id);
    }

    // Delete a book by ID
    public void delete(Integer id) {
        books.remove(id);
    }
}

```

3. **Title:** `BookService.java (Service interface)`

```java
package com.books_catelog.main.service;

import com.books_catelog.main.entity.Book;
import java.util.Map;
/** Defines our “business logic” operations. */
public interface BookService {

    Book addBook(Book book);

    Map<Integer, Book> getAllBooks(); // New method to retrieve all books

    Book getBookById(Integer id);

    void deleteBook(Integer id);

    void updateAvailability(Integer id, Boolean available);
}

```

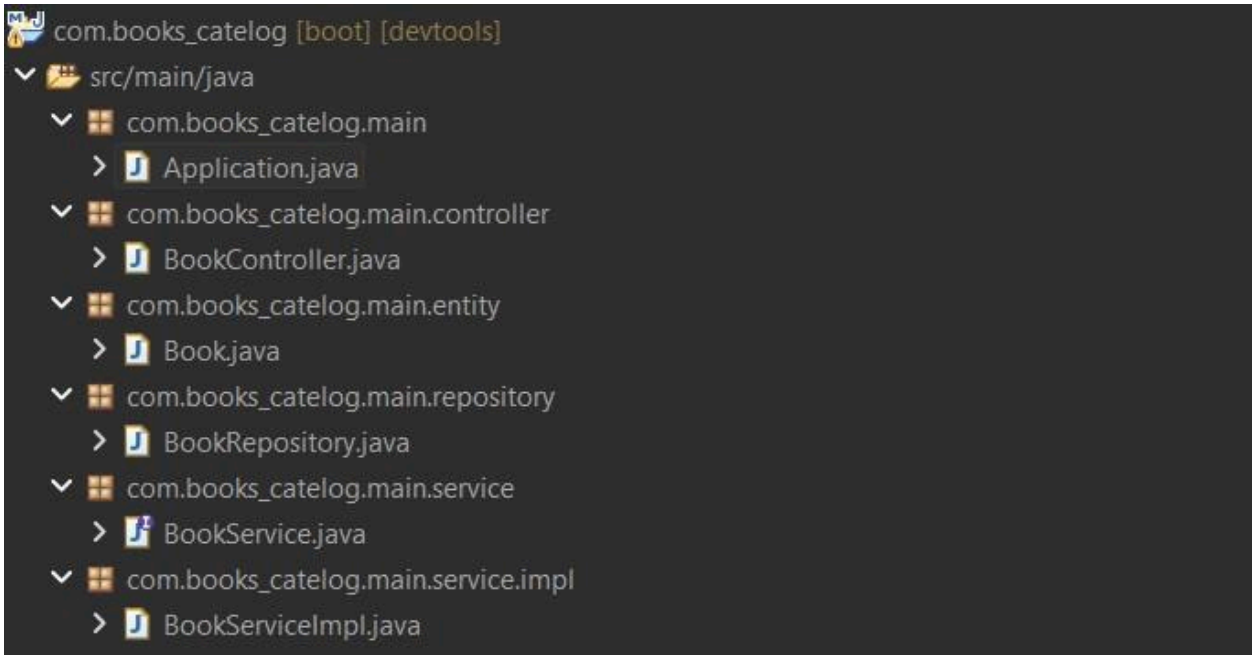
4. **Title:** `BookServiceImpl.java (Service Implementation class)`

```java
package com.books_catelog.main.service.impl;

import com.books_catelog.main.entity.Book;
import com.books_catelog.main.repository.BookRepository;
import com.books_catelog.main.service.BookService;
import org.springframework.stereotype.Service;

import java.util.Map;
/** Implements business rules on top of the repository. */
@Service
public class BookServiceImpl implements BookService {

    private final BookRepository bookRepository;

    public BookServiceImpl(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    @Override
    public Book addBook(Book book) {
        return bookRepository.save(book);
    }

    @Override
    public Map<Integer, Book> getAllBooks() {
        return bookRepository.findAll();
    }

    @Override
    public Book getBookById(Integer id) {
        return bookRepository.findById(id);
    }

    @Override
    public void deleteBook(Integer id) {
        bookRepository.delete(id);
    }

    @Override
    public void updateAvailability(Integer id, Boolean available) {
        Book book = bookRepository.findById(id);
        if (book != null) {
            book.setAvailable(available);
            bookRepository.save(book);
        }
    }
}

```

5. **Title:** `application properties`

```
# Basic Spring Boot configuration

# Server Configuration
server.port=8080
server.servlet.context-path=/api

# Enable debug mode to see detailed error messages
debug=true

# Enable detailed error page (useful in development)
spring.mvc.throw-exception-if-no-handler-found=true
spring.web.resources.add-mappings=false

# Logging Configuration
logging.level.org.springframework.web=DEBUG
logging.level.org.springframework.boot=DEBUG

```

6. Title: `Packaging Structure and Application.java (Main class)`



Application.java

```java
package com.books_catelog.main;

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

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

```

6. Title: Output Spring Boot Application on Spring tool suit