

# 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 -

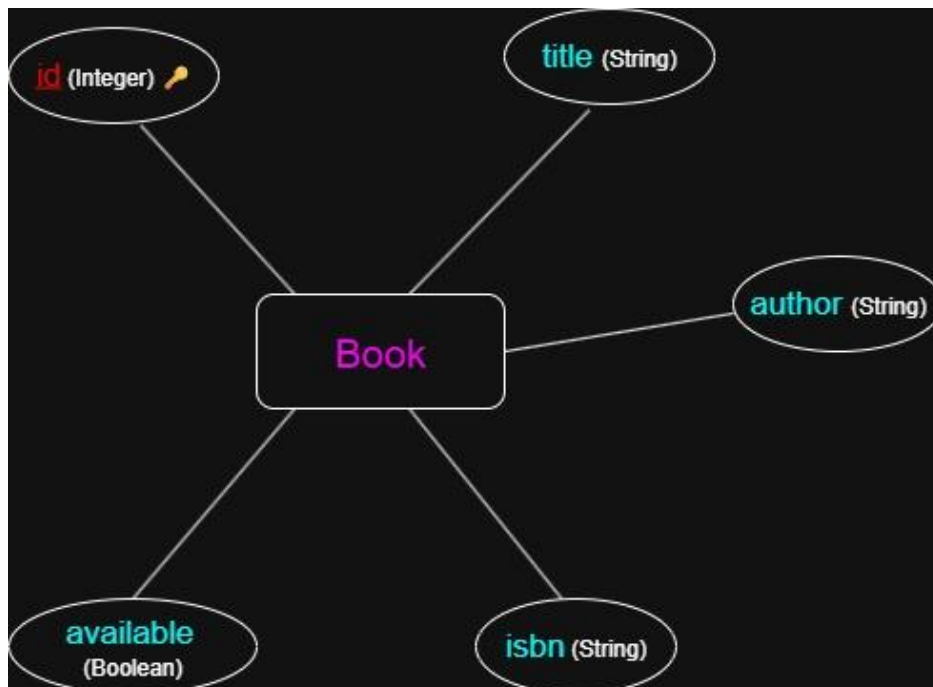
[https://github.com/darshanchaudhari/Library\\_Management\\_Books\\_Catalog\\_Backend\\_Server-.git](https://github.com/darshanchaudhari/Library_Management_Books_Catalog_Backend_Server-.git)

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)

```
1 package com.books_catelog.main.entity;
2
3 import java.util.concurrent.atomic.AtomicInteger;
4
5 import jakarta.validation.constraints.NotBlank;
6
7 /*
8  * Represents one Book in our catalog.
9  */
10 public class Book {
11     private static final AtomicInteger COUNTER = new AtomicInteger();
12     // unique identifier
13     private Integer id;
14     // If someone tries to create a book without a title validation will fail automatically
15     @NotBlank(message = "Title is required")
16     private String title;
17     // If someone tries to create a book without the author name validation will fail automatically
18     @NotBlank(message = "Author is required")
19     private String author;
20
21     private String isbn;
22
23     // Default value is true
24     private Boolean available = true;
25
26     // Constructor
27     public Book(String title, String author, String isbn) {
28         // Generates unique ID using AtomicInteger
29         this.id = COUNTER.incrementAndGet();
30         this.title = title;
31         this.author = author;
32         this.isbn = isbn;
33     }
34
35     // Getters and setters
36     public Integer getId() {
37         return id;
38     }
39
40     public void setId(Integer id) {
41         this.id = id;
42     }
43
44     public String getTitle() {
45         return title;
46     }
47
48     public void setTitle(String title) {
49         this.title = title;
50     }
51
52     public String getAuthor() {
53         return author;
54     }
55
56     public void setAuthor(String author) {
57         this.author = author;
58     }
59
60     public String getIsbn() {
61         return isbn;
62     }
63
64     public void setIsbn(String isbn) {
65         this.isbn = isbn;
66     }
67
68     public Boolean getAvailable() {
69         return available;
70     }
71
72     public void setAvailable(Boolean available) {
73         this.available = available;
74     }
75 }
76
```

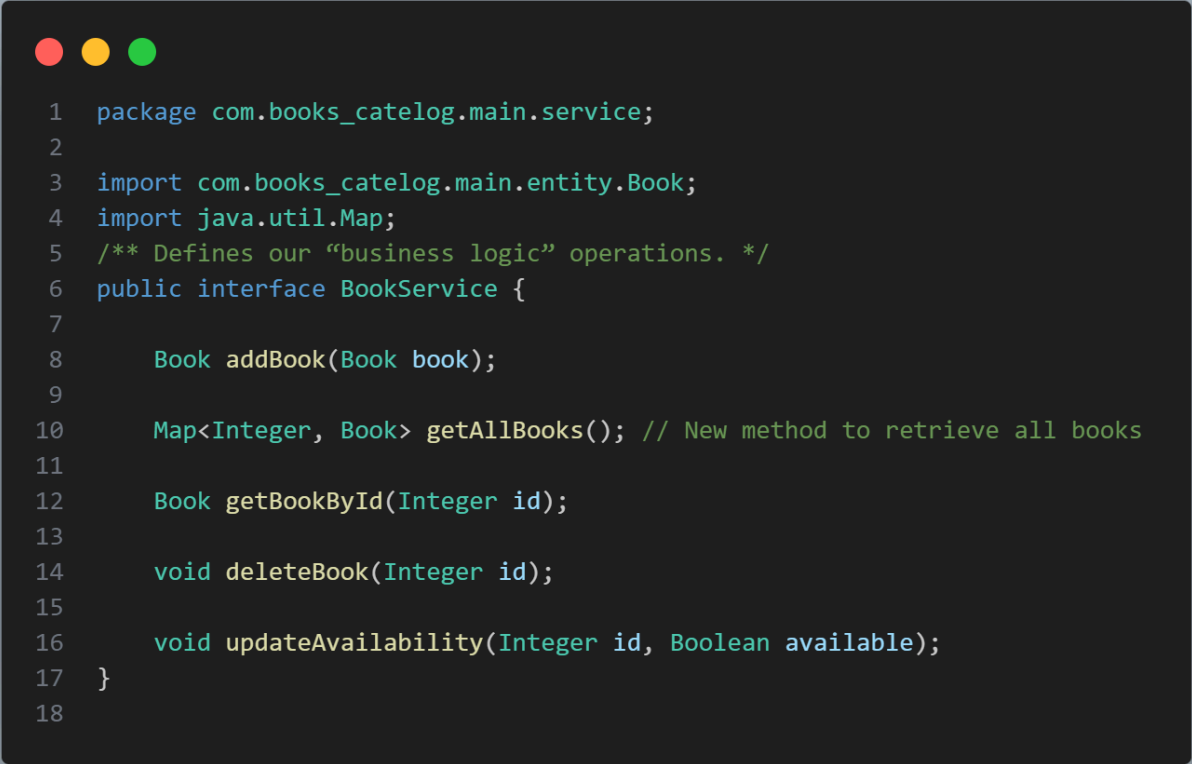
## 2. Title: BookController.java (Controller Class)

```
1 package com.books_catelog.main.controller;
2
3 import com.books_catelog.main.entity.Book;
4 import com.books_catelog.main.service.BookService;
5 import org.springframework.http.ResponseEntity;
6 import org.springframework.web.bind.annotation.*;
7
8 import java.util.Map;
9
10 /*
11  * Simple console menu to manage books in the library.
12  */
13 @RestController
14 @RequestMapping("/books")
15 public class BookController {
16
17     private final BookService bookService;
18
19     public BookController(BookService bookService) {
20         this.bookService = bookService;
21     }
22
23     // Add a new book
24     @PostMapping
25     public ResponseEntity<Book> addBook(@jakarta.validation.Valid @RequestBody Book book) {
26         Book savedBook = bookService.addBook(book);
27         return ResponseEntity.ok(savedBook);
28     }
29
30     // Get all books
31     @GetMapping
32     public ResponseEntity<Map<Integer, Book>> getAllBooks() {
33         Map<Integer, Book> books = bookService.getAllBooks();
34         return ResponseEntity.ok(books); // Returns the list of all books
35     }
36
37     // Get a book by ID
38     @GetMapping("/{id}")
39     public ResponseEntity<Book> getBookById(@PathVariable Integer id) {
40         Book book = bookService.getBookById(id);
41         return (book != null) ? ResponseEntity.ok(book) : ResponseEntity.notFound().build();
42     }
43
44     // Delete a book by ID
45     @DeleteMapping("/{id}")
46     public ResponseEntity<Void> deleteBook(@PathVariable Integer id) {
47         bookService.deleteBook(id);
48         return ResponseEntity.noContent().build();
49     }
50
51     // Update book availability
52     @PatchMapping("/{id}/availability")
53     public ResponseEntity<Void> updateAvailability(@PathVariable Integer id, @RequestParam Boolean available) {
54         bookService.updateAvailability(id, available);
55         return ResponseEntity.noContent().build();
56     }
57 }
58
```

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

```
1 package com.books_catelog.main.repository;
2
3 import com.books_catelog.main.entity.Book;
4 import org.springframework.stereotype.Repository;
5
6 import java.util.Map;
7 import java.util.concurrent.ConcurrentHashMap;
8
9 /**
10  * Inmemory store for Book objects.
11  */
12 @Repository
13 public class BookRepository {
14     // holds books
15     private final Map<Integer, Book> books = new ConcurrentHashMap<>();
16
17     // Save a book in the repository
18     public Book save(Book book) {
19         books.put(book.getId(), book);
20         return book;
21     }
22
23     // Get all books from the repository
24     public Map<Integer, Book> findAll() {
25         return books;
26     }
27
28     // Find a book by its ID
29     public Book findById(Integer id) {
30         return books.get(id);
31     }
32
33     // Delete a book by ID
34     public void delete(Integer id) {
35         books.remove(id);
36     }
37 }
38
```

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



```
1  package com.books_catelog.main.service;
2
3  import com.books_catelog.main.entity.Book;
4  import java.util.Map;
5  /** Defines our "business logic" operations. */
6  public interface BookService {
7
8      Book addBook(Book book);
9
10     Map<Integer, Book> getAllBooks(); // New method to retrieve all books
11
12     Book getBookById(Integer id);
13
14     void deleteBook(Integer id);
15
16     void updateAvailability(Integer id, Boolean available);
17 }
18
```

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

```
1 package com.books_catelog.main.service.impl;
2
3 import com.books_catelog.main.entity.Book;
4 import com.books_catelog.main.repository.BookRepository;
5 import com.books_catelog.main.service.BookService;
6 import org.springframework.stereotype.Service;
7
8 import java.util.Map;
9 /** Implements business rules on top of the repository. */
10 @Service
11 public class BookServiceImpl implements BookService {
12
13     private final BookRepository bookRepository;
14
15     public BookServiceImpl(BookRepository bookRepository) {
16         this.bookRepository = bookRepository;
17     }
18
19     @Override
20     public Book addBook(Book book) {
21         return bookRepository.save(book);
22     }
23
24     @Override
25     public Map<Integer, Book> getAllBooks() {
26         return bookRepository.findAll();
27     }
28
29     @Override
30     public Book getBookById(Integer id) {
31         return bookRepository.findById(id);
32     }
33
34     @Override
35     public void deleteBook(Integer id) {
36         bookRepository.delete(id);
37     }
38
39     @Override
40     public void updateAvailability(Integer id, Boolean available) {
41         Book book = bookRepository.findById(id);
42         if (book != null) {
43             book.setAvailable(available);
44             bookRepository.save(book);
45         }
46     }
47 }
48
```

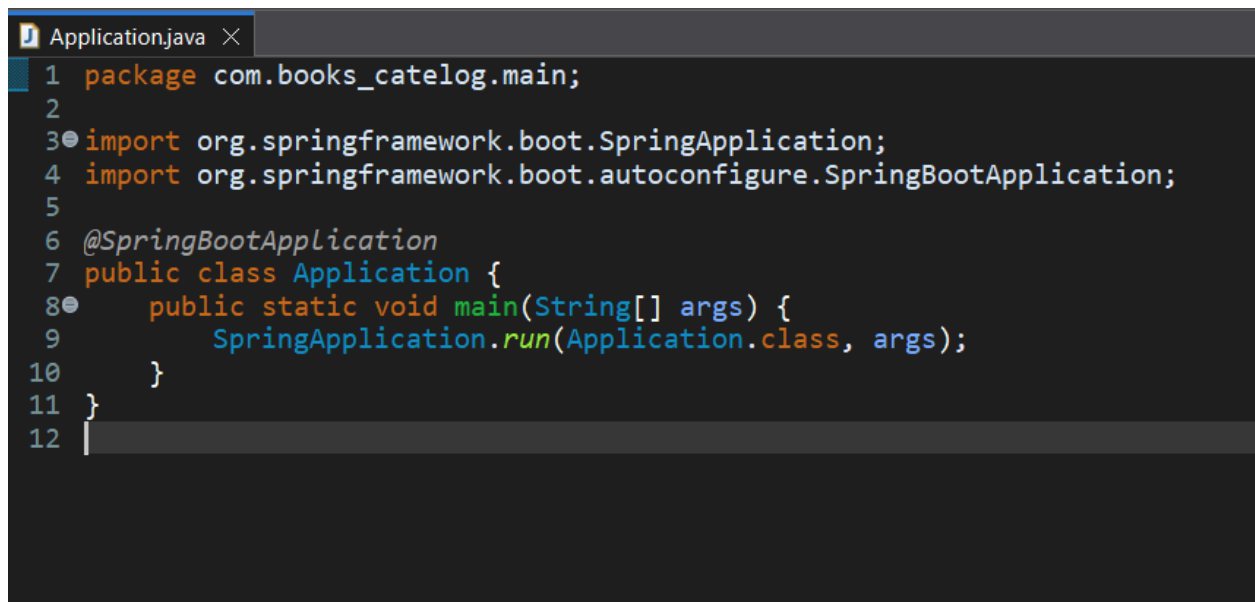
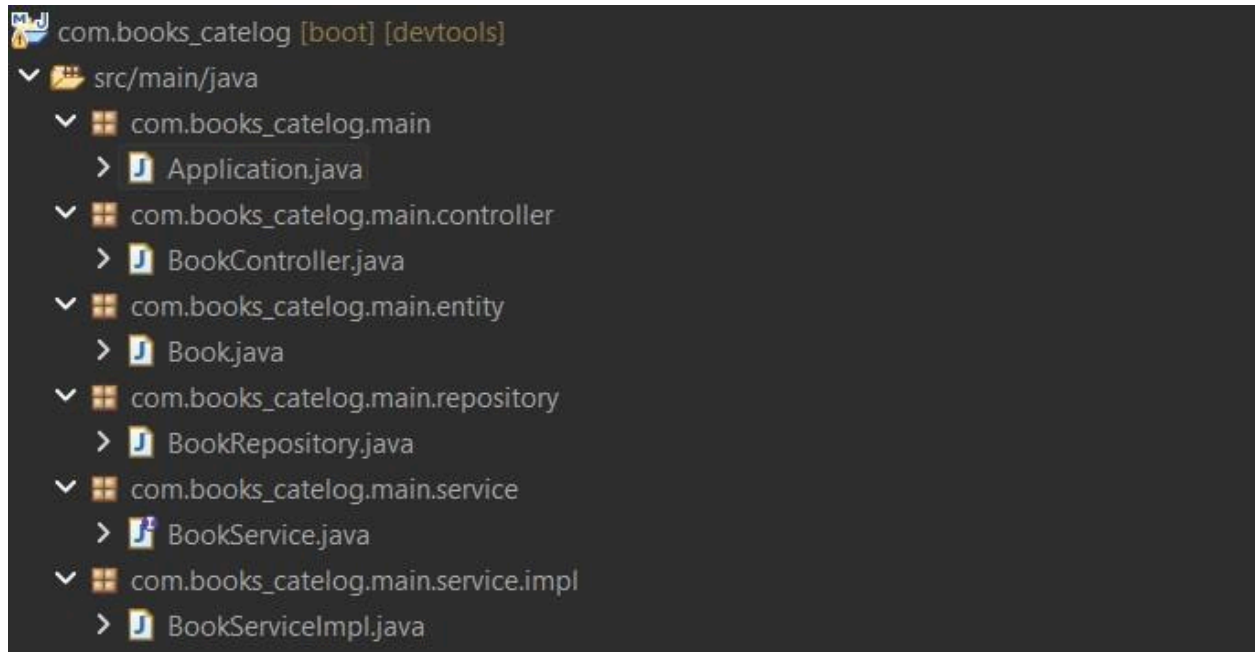
## 5. Title: application properties



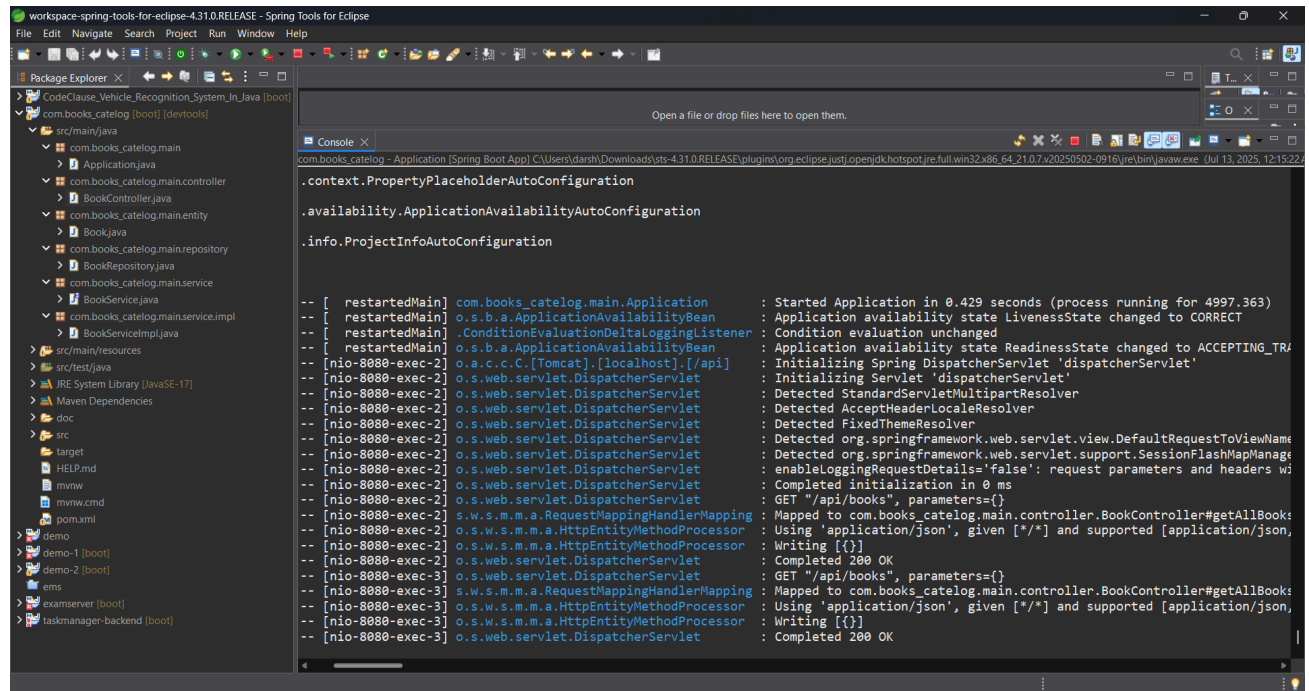
```
1  # Basic Spring Boot configuration
2
3  # Server Configuration
4  server.port=8080
5  server.servlet.context-path=/api
6
7  # Enable debug mode to see detailed error messages
8  debug=true
9
10 # Enable detailed error page (useful in development)
11 spring.mvc.throw-exception-if-no-handler-found=true
12 spring.web.resources.add-mappings=false
13
14 # Logging Configuration
15 logging.level.org.springframework.web=DEBUG
16 logging.level.org.springframework.boot=DEBUG
17
```



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



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



The screenshot displays the Eclipse IDE interface with the 'workspace-spring-tools-for-eclipse-4.31.0.RELEASE' project open. The Package Explorer on the left shows the project structure, including the 'com.books\_catalog' package and its sub-packages like 'src/main/java' and 'src/test/java'. The Console window on the right shows the output of the application, which is a Spring Boot application. The output includes the following lines:

```
.context.PropertyPlaceholderAutoConfiguration
.availability.ApplicationAvailabilityAutoConfiguration
.info.ProjectInfoAutoConfiguration

-- [ restartedMain] com.books_catalog.main.Application : Started Application in 0.429 seconds (process running for 4997.363)
-- [ restartedMain] o.s.b.a.ApplicationAvailabilityBean : Application availability state LivenessState changed to CORRECT
-- [ restartedMain] .ConditionEvaluationDeltaLoggingListener : Condition evaluation unchanged
-- [ restartedMain] o.s.b.a.ApplicationAvailabilityBean : Application availability state ReadinessState changed to ACCEPTING_TRA
-- [nio-8080-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/api] : Initializing Spring DispatcherServlet 'dispatcherServlet'
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Detected StandardServletMultipartResolver
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Detected AcceptHeaderLocaleResolver
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Detected FixedThemeResolver
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Detected org.springframework.web.servlet.view.DefaultRequestToViewName
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Detected org.springframework.web.servlet.support.SessionFlashMapManag
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : enableLoggingRequestDetails='false': request parameters and headers wi
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 0 ms
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : GET "/api/books", parameters={}
-- [nio-8080-exec-2] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to com.books_catalog.main.controller.BookController#getAllBooks
-- [nio-8080-exec-2] o.s.w.s.m.m.a.HttpEntityMethodProcessor : Using 'application/json', given [*/] and supported [application/json,
-- [nio-8080-exec-2] o.s.web.servlet.DispatcherServlet : Writing [{}]
```