

**A
Presentation On
Book Store Management System**

Presented by:

- 1.Jambhale Dipali(EBEON0522602992)
- 2.Vaidya Pratiksha(EBEON0522601614)
- 3.Tupake Pallavi(EBEON0522601621)
- 4.Nehe Shreya(EBTSOC0522598195)
- 5.Arakhade Prajakta(EBEON0522601543)



**Java Full Stack Developer
2022-7469**

Guided by:
Mehta Pooja mam

Contents

- Introduction
- **Required Specification**
- Proposed System
- Spring Annotation
- Usecase Diagram
- Class Diagram
- ER Diagram
- Screenshots
- Advantages

Online Book Store



Introduction

- Book Store Management System is designed to make the everyday working of the book store very easy and simple.
- It can be done wherever we are by just using the application.
- It keeps track of add the book , update the books , insert the new books, delete the book in the book store.
- It helps to provide all the information to the user .

Required Specification

Hardware Configuration	Software Configuration
Operating System : Windows 10,11	Software IDE : Spring Boot
Hard Disk : 1TB	Language : Java
RAM : 8GB	Back End : MySQL Server , Postman

Proposed System

- This system is a bunch of benefits from various point of view. As this book store management system enables the add different books in the book store ,update the books, delete the book , insert the new books in the book store.
- System will store the information about the books records.it will keep the records of each and every sold books and many more data.

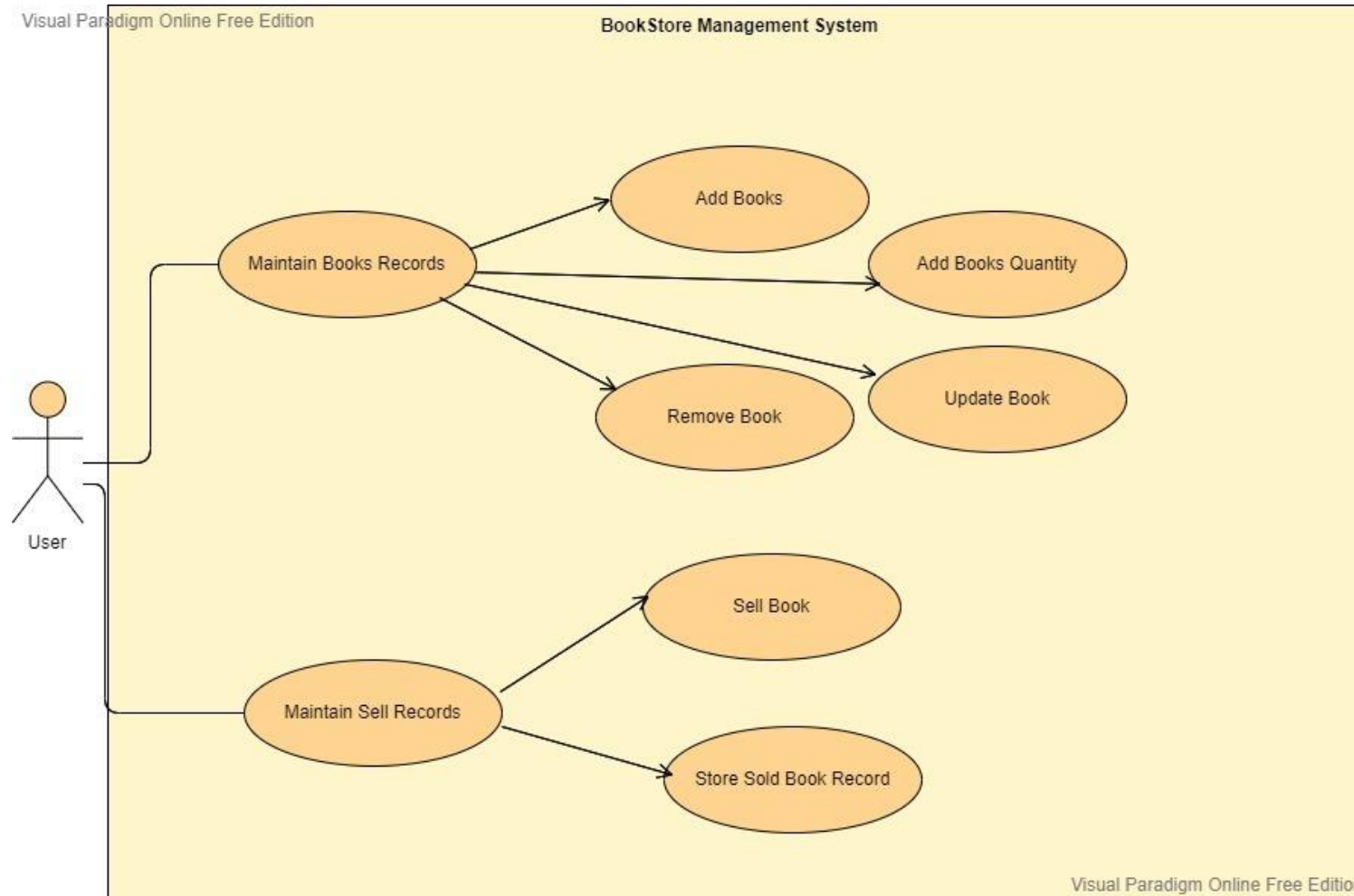
Spring Annotations

- `@Autowired`: The `@Autowired` annotation provide more fine grained control over where and how autowiring should be accomplished.
- `@RequestMapping` : The `@RequestMapping` is one of the most common annotation used in spring web application. This annotation maps HTTP requests to handler method of MVC and REST controllers.
- `@PathVariable` : The `@PathVariable` Spring annotation on a method argument to bind it to the value of a URI Template variable.
- `@RequestParam` : The bind Request Parameter to method variable using spring annotation `@RequestParam` .

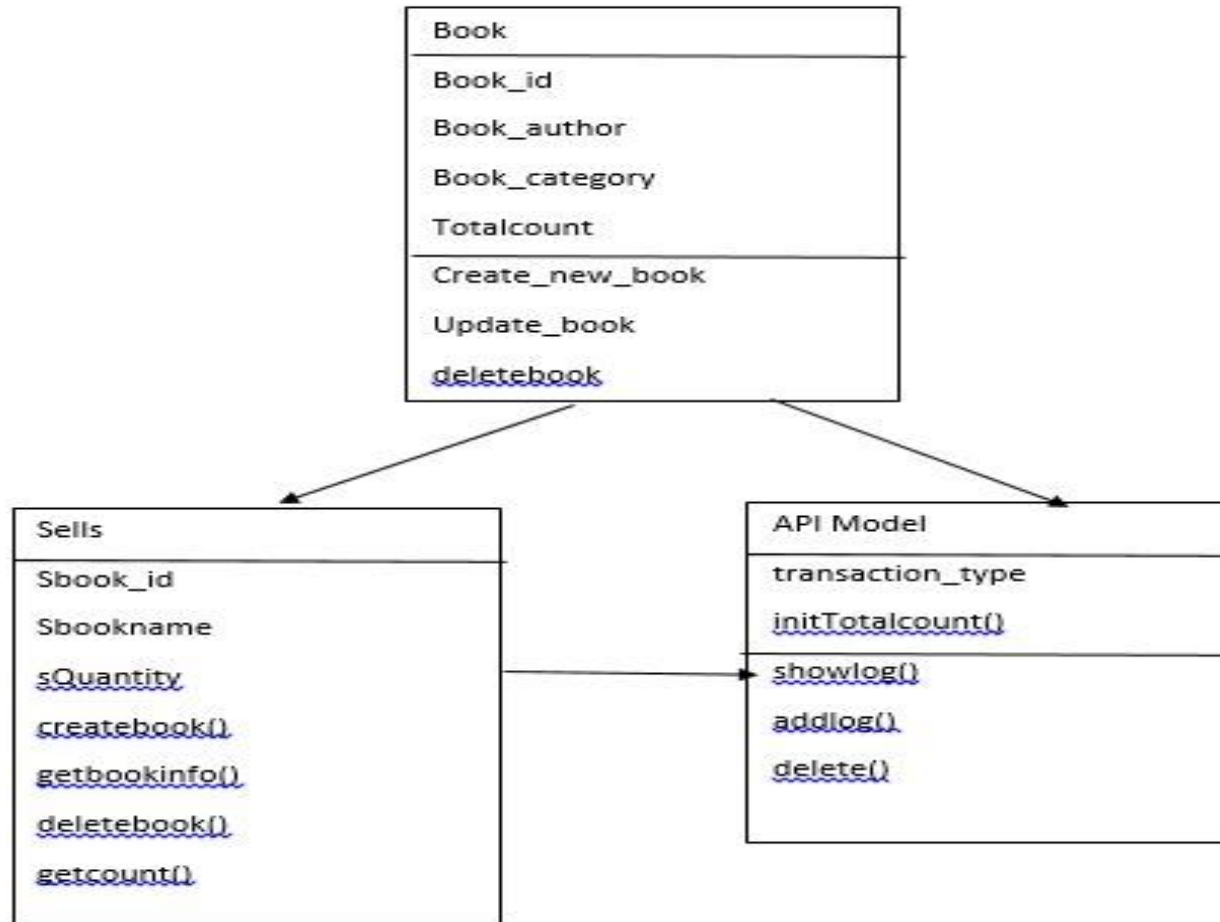
Spring Annotations

- @Configuration : Used to indicate that a class declares one or more @Bean methods . These classes are processed by the spring container to generate bean definition and service requests for those bean at runtime.
- @Bean : indicates that a method produces a bean to be managed by the spring container.
- @Controller : the @Controller annotation is used to indicate the class is a spring controller. This annotation can be used to identify controllers for spring MVC or Spring WebFlux .

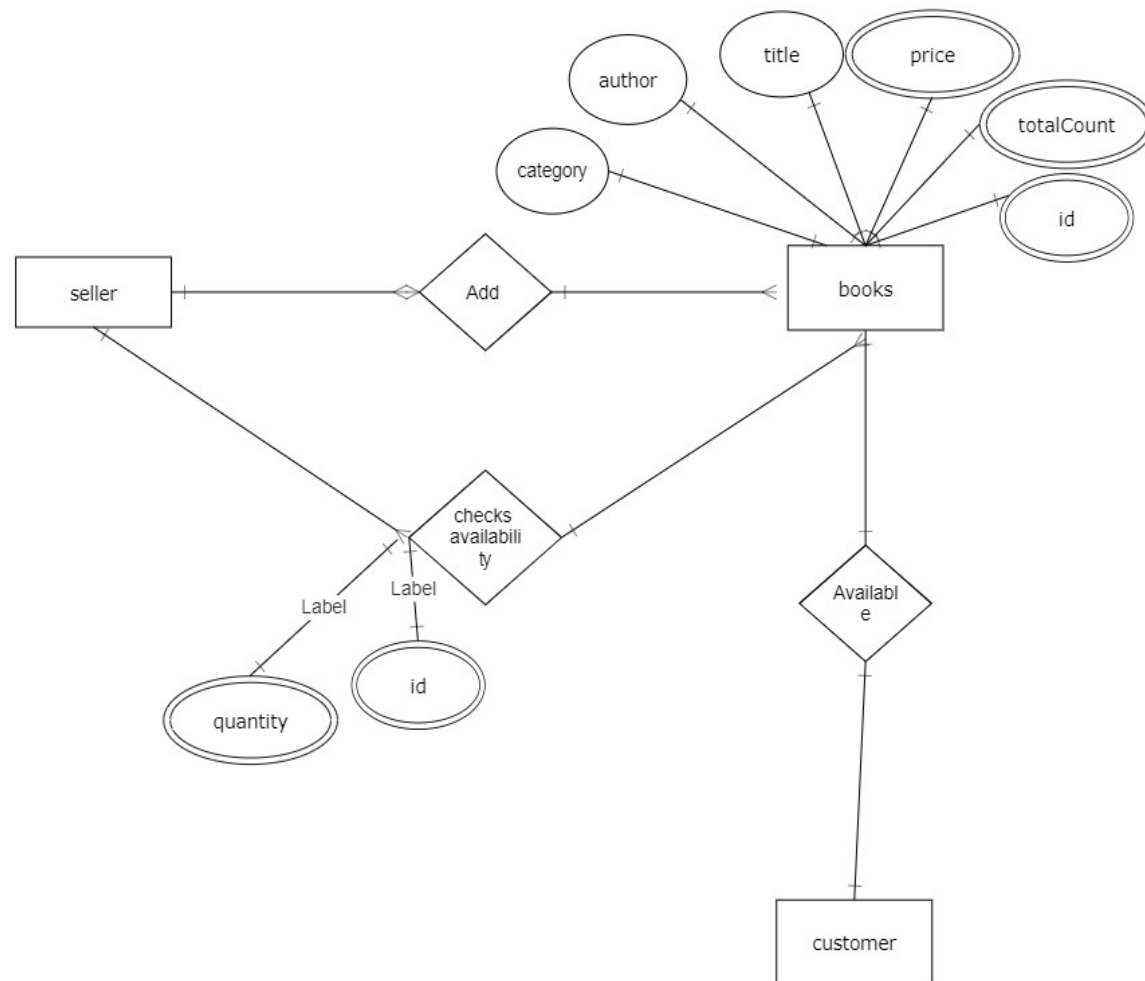
Usecase Diagram



Class Diagram

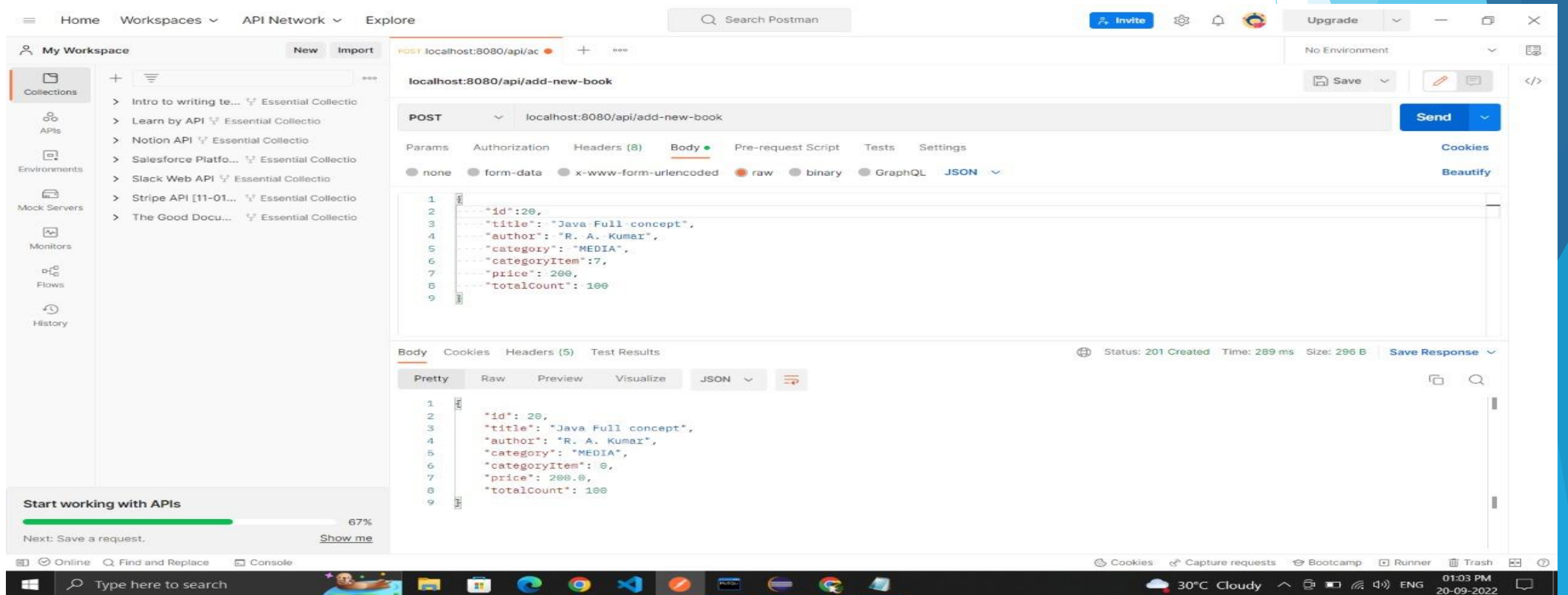


ER Diagram



Screenshots:

1] POST: Add New Books



2] PUT :

Add quantity of book to the existing book.

The screenshot displays the Postman interface for a PUT request. The URL is `localhost:8080/api/add-book/15/25`. The request body is a JSON object:

```
{
  "id": 20,
  "title": "Java Full concept",
  "author": "R. A. Kumar",
  "category": "MEDIA",
  "categoryItem": 7,
  "price": 200,
  "totalCount": 100
}
```

The response is a 200 OK status with a JSON object:

```
{
  "id": 15,
  "title": "Core Java ",
  "author": "Dr R. Nageswara Rao",
  "category": "LITERATURE",
  "price": 425.0,
  "totalCount": 125,
  "sold": 0
}
```

A watermark for SHREYA NEHE is visible on the right side of the image.

2] PUT :

Update a book.

The screenshot displays the Postman API client interface. The top navigation bar includes 'Home', 'Workspaces', 'API Network', and 'Explore'. The left sidebar shows 'My Workspace' with a 'Collections' list containing various API collections like 'Intro to writing te...', 'Learn by API', 'Notion API', 'Salesforce Platfo...', 'Slack Web API', 'Stripe API', and 'The Good Docu...'. The main workspace shows a PUT request to 'localhost:8080/api/books/15'. The 'Body' tab is selected, showing a JSON payload:

```
{  "id": 17,  "title": "Java Full concept",  "author": "R. A. Kumar",  "category": "THRILLER",  "categoryItem": 3,  "price": 200,  "totalCount": 100}
```

. The bottom section shows the 'Body' tab with a 'Pretty' view of the JSON response:

```
{  "id": 15,  "title": "Java Full concept",  "author": "R. A. Kumar",  "category": "THRILLER",  "price": 200.0,  "totalCount": 100,  "sold": 0,  "hibernateLazyInitializer": {}}
```

. The status bar at the bottom indicates 'Status: 200 OK', 'Time: 83 ms', and 'Size: 316 B'. The Windows taskbar at the very bottom shows the system clock as 01:53 PM on 20-09-2022, along with weather information (27°C Rain showers) and various application icons.

2]PUT

Sell a single book

The screenshot displays the Postman API client interface. The top navigation bar includes 'Home', 'Workspaces', 'API Network', and 'Explore'. The left sidebar shows 'My Workspace' with options for 'New' and 'Import'. The main area is titled 'localhost:8080/api/sell-book/3' and shows a 'PUT' request. The request body is a JSON object:

```
{  "id": 6,  "title": "OS-",  "author": "P.S.GAIKWAD",  "price": 106,  "totalCount": 1180}
```

. The response status is '200 OK' with a time of '157 ms' and a size of '123 B'. The response body is displayed in 'Pretty' format, showing the number '1'. The bottom status bar shows '80°F Cloudy' and the system clock '3:53 PM 9/20/2022'.

Home Workspaces API Network Explore

Search Postman

Invite Upgrade

My Workspace New Import Overview

localhost:8080/api/se

localhost:8080/api/sell-book/3

PUT localhost:8080/api/sell-book/3

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "id": 6,
3   "title": "OS-",
4   "author": "P.S.GAIKWAD",
5   "price": 106,
6   "totalCount": 1180
7 }
```

Body Cookies Headers (4) Test Results

Status: 200 OK Time: 157 ms Size: 123 B Save Response

Pretty Raw Preview Visualize Text

1

Online Find and Replace Console

80°F Cloudy

3:53 PM 9/20/2022

3] GET:

Get books by id

The screenshot displays the Postman application interface. At the top, a green status bar indicates 'Remaining Meeting Time: 05:03'. The main workspace shows a GET request to 'localhost:8080/api/book/17'. The 'Body' tab is selected, showing a JSON response with the following structure:

```
1 {
2   "id": 17,
3   "title": "Java Full concept",
4   "author": "R. A. Kumar",
5   "category": "THRILLER",
6   "categoryItem": 3,
7   "price": 200,
8   "totalCount": 100
9 }
```

Below the request, the 'Body' tab of the response is selected, showing the same JSON data in a 'Pretty' format:

```
1 {
2   "id": 17,
3   "title": "Java Full concept",
4   "author": "R. A. Kumar",
5   "category": "MEDIA",
6   "categoryItem": 0,
7   "price": 200.0,
8   "totalCount": 100
9 }
```

The status bar at the bottom indicates 'Status: 200 OK', 'Time: 58 ms', and 'Size: 291 B'. A 'Start working with APIs' progress bar at the bottom left shows 67% completion.

3] GET

Get All Books

The screenshot shows the Postman interface with a GET request to `localhost:8080/api/book-list` executed successfully. The response is a JSON array of three book objects. The status is 200 OK, with a response time of 395 ms and a size of 1.47 KB.

Request:

- Method: GET
- URL: `localhost:8080/api/book-list`

Response Body (JSON):

```
[{"id": 7, "title": "Introduction to Algorithms", "author": "Thomas H. Cormen", "category": "LITERATURE", "categoryItem": 0, "price": 0.0, "totalCount": 0}, {"id": 8, "title": "Getting inside Java", "author": "Prem Kumar", "category": "NONFICTION", "categoryItem": 0, "price": 750.0, "totalCount": 44}, {"id": 9, "title": "Full Of Java ", "author": "Lalji Mehta", "category": "OTHERS", "categoryItem": 0, "price": 650.0, "totalCount": 14}]
```

Start working with APIs

67% progress bar

Next: Save a request. [Show me](#)

3] GET :

Get number of books available by id.

The screenshot displays the Postman API client interface. At the top, there's a navigation bar with 'Home', 'Workspaces', 'API Network', and 'Explore'. A green notification bar indicates 'Remaining Meeting Time: 01:41'. The main workspace shows a 'My Workspace' sidebar with various collections and environments. The central panel is set to a 'GET' request to 'localhost:8080/api/number-of-books/17'. The 'Body' tab is selected, showing a JSON response:

```
1 {
2   "id": 17,
3   "title": "Java Full concept",
4   "author": "R. A. Kumar",
5   "category": "THRILLER",
6   "categoryItem": 3,
7   "price": 200,
8   "totalCount": 100
9 }
```

Below the request, the 'Body' tab of the response is selected, showing the same JSON object in a 'Pretty' format:

```
1 {
  "id": 17,
  "title": "Java Full concept",
  "author": "R. A. Kumar",
  "category": "THRILLER",
  "categoryItem": 3,
  "price": 200,
  "totalCount": 100
}
```

The status bar at the bottom indicates 'Status: 200 OK', 'Time: 28 ms', and 'Size: 167 B'. A 'Start working with APIs' progress bar shows 67% completion.

4] DELETE :

Delete the record by using id.

The screenshot displays the Postman application interface. On the left sidebar, the 'My Workspace' section is active, showing a list of collections. The main panel is configured for a DELETE request to the endpoint `localhost:8080/api/book/11`. The request body is set to 'raw' and contains the following JSON data:

```
1 {
2   "id": 17,
3   "title": "Java Full concept",
4   "author": "R. A. Kumar",
5   "category": "THRILLER",
6   "categoryItem": 3,
7   "price": 200,
8   "totalCount": 100
9 }
```

The response section at the bottom shows a status of 200 OK, a time of 40 ms, and a size of 170 B. The response body is displayed in 'Pretty' format as:

```
1 success
```

At the bottom of the screen, the Windows taskbar is visible, showing the system clock at 02:00 PM on 20-09-2022, along with weather information (27°C Rain showers) and various system icons.

```
mysql> show databases;
```

Database
bookstore
ebs
edubridge
electricity
hospital
information_schema
manifest
mysql
mytestdb
performance_schema
sys
test

```
12 rows in set (0.01 sec)
```

```
mysql> use bookstore;
```

```
Database changed
```

```
mysql> show tables;
```

Tables_in_bookstore
book

```
1 row in set (0.01 sec)
```

```
mysql> desc book;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
author	varchar(255)	YES		NULL	
category	int	YES		NULL	
price	float	NO		NULL	
sold	int	NO		NULL	
title	varchar(255)	YES		NULL	
total_count	int	NO		NULL	

```
7 rows in set (0.01 sec)
```

```
mysql> select *from book;
```

id	author	category	price	sold	title	total_count
1	P.S.GAIKWAD	0	150	0	PYTHON	1330

```
manifest
mysql
mytestdb
performance_schema
sys
test
```

12 rows in set (0.01 sec)

```
mysql> use bookstore;
```

Database changed

```
mysql> show tables;
```

```
Tables_in_bookstore |
```

```
book |
```

1 row in set (0.01 sec)

```
mysql> desc book;
```

Field	Type	Null	Key	Default	Extra
id	int	NO	PRI	NULL	auto_increment
author	varchar(255)	YES		NULL	
category	int	YES		NULL	
price	float	NO		NULL	
sold	int	NO		NULL	
title	varchar(255)	YES		NULL	
total_count	int	NO		NULL	

7 rows in set (0.01 sec)

```
mysql> select *from book;
```

id	author	category	price	sold	title	total_count
1	P.S.GAIKWAD	0	150	0	PYTHON	1330
2	P.A.VAIDYA	0	120	0	RUBY	130
3	P.A.VAIDYA	0	120	1	RUBY	129
4	N.R.JADHAV	0	100	0	java	1130
5	M.P.SHELKE	0	106	0	OOP	1180
6	P.S.GAIKWAD	0	106	0	OS	1180

6 rows in set (0.00 sec)

```
mysql> _
```

Advantages

- It is User-friendly System.
- It keeps tracks of all the books.
- It gives us all the information about the books.
- It focuses on day to day operations in book store such as the user search book by using id.
- It increases efficiency in managing the books .

Thank You...