**Exercise 2: Online Bookstore - Creating Basic REST Controllers**

1. Creating the BookController Class:

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import org.springframework.web.bind.annotation.\*;

import java.util.ArrayList;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

private List<Book> bookList = new ArrayList<>();

// 2.1 Handle GET requests (retrieve all books)

@GetMapping

public List<Book> getAllBooks() {

return bookList;

}

// 2.2 Handle GET by ID (retrieve a specific book)

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookList.stream()

.filter(book -> book.getId().equals(id))

.findFirst()

.orElse(null);

}

// 2.3 Handle POST requests (add a new book)

@PostMapping

public Book createBook(@RequestBody Book newBook) {

bookList.add(newBook);

return newBook;

}

// 2.4 Handle PUT requests (update an existing book)

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

for (Book book : bookList) {

if (book.getId().equals(id)) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

book.setIsbn(updatedBook.getIsbn());

return book;

}

}

return null;

}

// 2.5 Handle DELETE requests (delete a book)

@DeleteMapping("/{id}")

public String deleteBook(@PathVariable Long id) {

bookList.removeIf(book -> book.getId().equals(id));

return "Book deleted";

}

}

1. Define the Book Entity:

package com.example.bookstoreapi.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Book {

private Long id;

private String title;

private String author;

private double price;

private String isbn;

}

**3. Returning JSON Responses**

By default, Spring Boot's @RestController automatically serializes Java objects (like the Book entity) to JSON format, so there’s no additional configuration needed. The methods in BookController will automatically return JSON responses for GET, POST, PUT, and DELETE requests.

**Endpoints and Example Requests**

* **GET** /books: Fetches all books.
* **GET** /books/{id}: Fetches a book by its ID.
* **POST** /books: Adds a new book.

**Example Request Body:**

{

"id": 1,

"title": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"price": 10.99,

"isbn": "9780743273565"

}

**PUT** /books/{id}: Updates a book by its ID.

**Example Request Body:**

{

"title": "The Great Gatsby",

"author": "F. Scott Fitzgerald",

"price": 11.99,

"isbn": "9780743273565"

}

* **DELETE** /books/{id}: Deletes a book by its ID.

This basic implementation allows us to manage books using simple RESTful services. We can test the API with tools like Postman or using curl commands.