**WEEK 4&5**

**EXERCISE 2**

**Online Bookstore - Creating Basic REST Controllers**

**Business Scenario**

The task is to implement RESTful endpoints to manage books in the online bookstore application.

**Instructions**

**1. Create Book Controller:**

Step 1: Define a BookController class within the controller package.

* Request Mapping: Use the @RestController annotation to define the class as a REST controller.
* Endpoint: Use the @RequestMapping("/books") annotation to map HTTP requests to /books URL.

@RestController

@RequestMapping("/books")

public class BookController {

private final BookService bookService;

public BookController(BookService bookService) {

this.bookService = bookService;

}

**2. Handle HTTP Methods:**

Implement methods in BookController to handle the following HTTP methods:

* **GET:** Retrieve a list of books or a single book by ID.

@GetMapping

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) {

return bookService.getBookById(id)

.map(ResponseEntity::ok)

.orElse(ResponseEntity.notFound().build());

}

* **POST:** Add a new book to the collection.

@PostMapping

public ResponseEntity<Book> addBook(@RequestBody Book book) {

Book savedBook = bookService.saveBook(book);

return ResponseEntity.status(HttpStatus.CREATED).body(savedBook);

}

* **PUT:** Update an existing book's details.

@PutMapping("/{id}")

public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

return bookService.getBookById(id)

.map(existingBook -> {

existingBook.setTitle(bookDetails.getTitle());

existingBook.setAuthor(bookDetails.getAuthor());

existingBook.setPrice(bookDetails.getPrice());

existingBook.setIsbn(bookDetails.getIsbn());

Book updatedBook = bookService.saveBook(existingBook);

return ResponseEntity.ok(updatedBook);

})

.orElse(ResponseEntity.notFound().build());

}

* **DELETE:** Remove a book from the collection.

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

boolean isDeleted = bookService.deleteBook(id);

if (isDeleted) {

return ResponseEntity.noContent().build(); // HTTP 204 No Content

} else {

return ResponseEntity.notFound().build(); // HTTP 404 Not Found

}

}

**3. Return JSON Responses:**

**Step 1:**

Ensure that the BookController methods return JSON responses by default. Spring Boot automatically converts Java objects returned by the controller methods into JSON format.

**Step 2:**

Define the Book entity class with the following attributes:

* **id:** A unique identifier for each book (Long).
* **title:** The title of the book (String).
* **author:** The author of the book (String).
* **price:** The price of the book (double).
* **isbn:** The ISBN number of the book (String).

public class Book {

private Long id;

private String title;

private String author;

private double price;

private String isbn;

// Constructor

public Book() {}

public Book(Long id, String title, String author, double price, String isbn) {

this.id = id;

this.title = title;

this.author = author;

this.price = price;

this.isbn = isbn;

}

// Getters and Setters

public Long getId() {

return id;

}

public void setId(Long 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 double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public String getIsbn() {

return isbn;

}

public void setIsbn(String isbn) {

this.isbn = isbn;

}

}

**Conclusion:**

By following these steps, a fully functional RESTful service to manage books, allowing CRUD (Create, Read, Update, Delete) operations via HTTP methods.