package com.example.bookstoreapi.model;

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Book {

private Long id;

private String title;

private String author;

private double price;

}

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Author {

private Long id;

private String name;

}

@Data

@NoArgsConstructor

@AllArgsConstructor

public class Customer {

private Long id;

private String name;

private String email;

}

Service Layer

package com.example.bookstoreapi.service;

import com.example.bookstoreapi.model.Book;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

private final List<Book> books = new ArrayList<>();

public List<Book> getAllBooks() {

return books;

}

public Optional<Book> getBookById(Long id) {

return books.stream().filter(book -> book.getId().equals(id)).findFirst();

}

public Book addBook(Book book) {

books.add(book);

return book;

}

public void deleteBook(Long id) {

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

}

}

Controller Classes

package com.example.bookstoreapi.controller;

import com.example.bookstoreapi.model.Book;

import com.example.bookstoreapi.service.BookService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookService bookService;

@GetMapping

public List<Book> getAllBooks() {

return bookService.getAllBooks();

}

@GetMapping("/{id}")

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

Optional<Book> book = bookService.getBookById(id);

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

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

Book savedBook = bookService.addBook(book);

return new ResponseEntity<>(savedBook, HttpStatus.CREATED);

}

@DeleteMapping("/{id}")

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

bookService.deleteBook(id);

**What's New in Spring Boot 3**

Here are some of the new features introduced in Spring Boot 3:

1. **Java 17 as the Baseline:**
   * Spring Boot 3 requires at least Java 17, taking advantage of the new language features and performance improvements.
2. **Improved GraalVM Native Support:**
   * Native image support has been improved, making it easier to build and deploy native executables.
3. **Observability Improvements:**
   * Enhanced support for Micrometer and OpenTelemetry, allowing for better observability and tracing in distributed systems.
4. **HTTP Interface Client:**
   * Introduces a new way to declare HTTP clients using annotated interfaces, simplifying REST client creation.
5. **Kubernetes Probes Integration:**
   * Better integration with Kubernetes health probes (liveness, readiness) out of the box.
6. **Docker Compose Support:**
   * Direct support for Docker Compose in development environments, making it easier to manage dependencies like databases during development.
7. **Enhanced Dependency Management:**
   * Spring Boot 3 provides enhanced dependency management, especially for projects using Gradle, with improved support for version alignment and BOM (Bill of Materials).
8. **WebSocket Improvements:**
   * Improved WebSocket support with better scalability and performance tuning options.
9. **Testcontainers Integration:**
   * Spring Boot 3 has built-in support for Testcontainers, making it easier to use containers in integration tests.
10. **Security Improvements:**
    * Updates and improvements in Spring Security, including better OAuth 2.0 support and new configuration options.